PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 13/00, 17/00

A1

(11) International Publication Number: **WO 00/42513**

(43) International Publication Date: 20 July 2000 (20.07.00)

(21) International Application Number: PCT/US00/00615

(22) International Filing Date: 11 January 2000 (11.01.00)

(30) Priority Data:
60/115,376 11 January 1999 (11.01.99) US

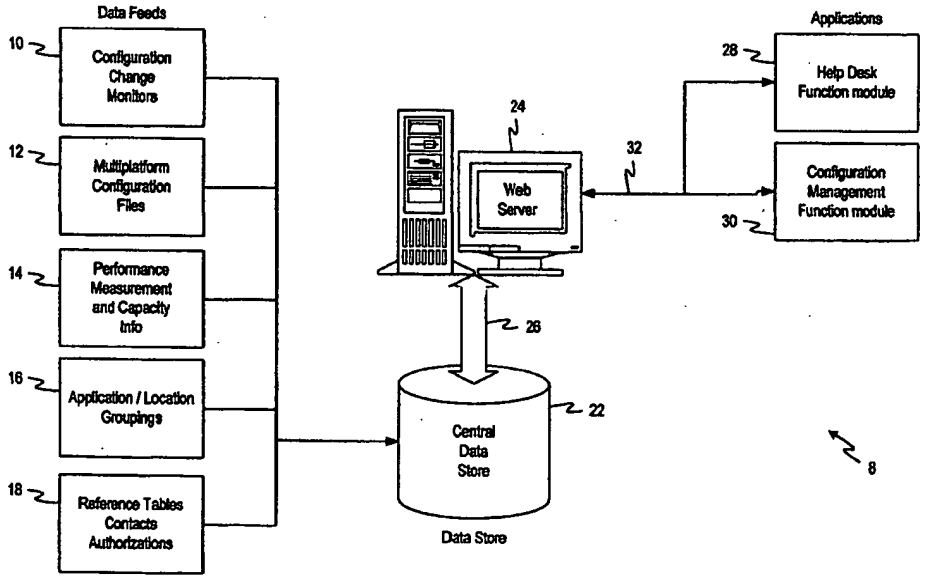
(71)(72) Applicant and Inventor: KUYKENDALL, William [US/US]; 2253 Highmoor Road, Highland Park, IL 60035 (US).

(74) Agent: RUDISILL, Stephen; Jenkens & Gilchrist, P.C., Suite 3200, 1445 Ross Avenue, Dallas, TX 75202-2799 (US).

(81) Designated States: AU, CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: NETWORK MANAGEMENT SYSTEM



(57) Abstract

A network management system utilizes a multipurpose data collection agent (200) to collect performance data, configuration and configuration change data, and other system data from computers and routers in a network. The multipurpose data collection agent (200) calls specific data collection subroutines to gather data from the data sources (14). The data is stored in a central data store (22), which is accessible by a web server (24). The data store is then used by a help desk function module to provide efficient and responsive solutions to existing or potential network problems.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

NETWORK MANAGEMENT SYSTEM

RELATED APPLICATIONS

This is a complete application claiming the benefit of co-pending Provisional Patent Application No. 60/115,376, filed January 11, 1999.

FIELD OF THE INVENTION

The invention relates generally to the field of information technology. In particular, the invention relates to the management of combinations of computers engaged in providing one or more integrated services.

BACKGROUND OF THE INVENTION

Most applications of computer technology used for the purposes of communications and data management in commercial and governmental applications are constructed using the computing model known as the client-server model. The client-server model is generally defined as two or more computers which communicate via a network, with each computer providing partial functionality in cooperation with the other parts to deliver a unified computer application or service. Each "computer" in the client-server model refers to one or more hardware devices, operating systems, and computer programs which perform a finite set of tasks. More detailed descriptions of the client-server computing model and the technologies employed in specific implementations can be found in *Computer Networks, 3d Edition* by Andrew S. Tanenbaum, *Unix Network Programming* by W. Richard Stevens, and *Networking Applications on UNIX System V Release 4* by Michael Padovano.

The goals of managing these client-server implementations include ensuring service availability, performance and recoverability through the range of problems which can occur within each implementation. Because of the complexity of these compound systems and the rapid pace of development within each component technology, information technology management tends to be organized in specialized groups dedicated to each component technology. The current generation of software

tools to aid in the management of these systems tends to be organized around these same divisions of labor.

This approach has many shortcomings. The early stages of problem identification and resolution are grossly inefficient inasmuch as problems are reported on a service or application basis rather than by component technology. Change management and disaster recovery effectiveness is limited by the sheer number of people who must participate in coordinated efforts to accomplish these goals. Because proactive monitoring capabilities are segmented by technology group, many problems are not recognized until there is a service outage. This increases the dependence on problem identification and resolution for business continuity, resulting in a reactive approach to systems management, which further degrades attempts at change management.

SUMMARY OF THE INVENTION

The primary purpose of this invention is to document changes to system configurations as they occur within all components of any complex computing environment.

Another object of this invention is to apply controls to these changes after they have occurred. An example of this function is the case where multiple machines provide a single application resource, such as a cluster of web servers. Configuration change controls are implemented through rules. One rule would require a given change to be propagated to all machines providing that single service, or require the change to be retracted from the server on which it was detected.

Another object of this invention is to analyze the effects of these changes, and identify which applications and services may be affected by them. This feature is crucial to prompt identification of service outage causes.

Another object of this invention is to provide perpetual documentation of system configurations (and copies of those configurations) for every point in time in recent history.

Another object of this invention is to provide historical data pertaining to resource utilization and system performance for the purpose of calculating trends and

projecting future capacity requirements, and providing service level agreement documentation, and employee effectiveness tracking.

Another object of the present invention is to provide a real-time representation of the general health of the systems being monitored and the effects of changes made to them by the operations staff.

In accordance with the present invention, these and other objectives are realized by a method of managing complex networks, said method comprising collecting and storing in a central data store configuration change information from data collection agents executing on said client computers; collecting and storing in a central data store multi-platform configuration files from said data collection agents; collecting and storing in a central data store performance measurement and capacity information from said data collection agents; collecting and storing in a central data store application and location grouping information from said data collection agents; collecting and storing in a central data store reference table information from said data collection agents; and providing said configuration change information, multi-platform configuration files, performance measurement and capacity information, application and location grouping information, and reference table information to a help desk function module and a configuration management function module.

In a preferred embodiment of the invention, configuration changes in the network are monitored by executing configuration data collection agents on one or more computers, with the data collection agents tracking and time-stamping configuration changes to said computers. The configuration data collection agents then report said configuration changes to a central data store.

BRIEF DESCRIPTION OF THE DRAWINGS

In the accompanying drawings:

FIG. 1 is a diagram giving a functional overview of one embodiment of the present invention;

FIG. 1a is a block diagram showing various types of data contained in a central data store;

FIG. 2 is flow diagram illustrating the operation of a multipurpose data collection agent for use in one embodiment of the present invention;

FIG. 3 is a flow diagram illustrating a TELNET emulation access method for use in one embodiment of the present invention;

FIG. 4 is a flow diagram illustrating a "transmit" subroutine for use in one embodiment of the present invention;

FIG. 5 is a flow diagram illustrating a "waitfor" subroutine for use in one embodiment of the present invention;

FIG. 6 is flow diagram illustrating a configuration change monitor subroutine for use in one embodiment of the present invention;

FIG. 7 is a flow diagram illustrating a web server performance monitor subroutine for use in one embodiment of the present invention;

FIG. 8 is a flow diagram illustrating router capacity and performance monitor subroutine for use in one embodiment of the present invention;

FIG. 9 is a flow diagram illustrating a router configuration change monitor subroutine for use in one embodiment of the present invention;

FIG. 10 is a flow diagram illustrating a generic capacity monitor subroutine for use in one embodiment of the present invention;

FIG. 11 is a flow diagram illustrating a customer database availability and performance monitor subroutine for use in one embodiment of the present invention;

FIG. 12 is a flow diagram illustrating a logic overview for a help desk function module for use in one embodiment of the present invention;

FIG. 13 is a flow diagram illustrating a ticket creation phase for a help desk function module for use in one embodiment of the present invention;

FIG. 14 is a flow diagram illustrating a ticket assignment phase for a help desk function module for use in one embodiment of the present invention;

FIG. 15 is a flow diagram illustrating a candidate cause identification subroutine for use in one embodiment of the present invention;

FIG. 16 is a flow diagram illustrating a ticket resolution phase for a help desk function module for use in one embodiment of the present invention;

FIG. 17 is a flow diagram illustrating a resolution or rejection CGI application for use in one embodiment of the present invention;

FIG. 18 is a flow diagram illustrating a logic overview for a configuration management function module for use in one embodiment of the present invention;

FIG. 19 is a flow diagram illustrating an approval request phase for a configuration management function module for use in one embodiment of the present invention;

FIG. 20 is a flow diagram illustrating a contact approval phase for a configuration management function module for use in one embodiment of the present invention;

FIG. 21 is a flow diagram illustrating an approval CGI program for use in one embodiment of the present invention;

FIG. 22 is a flow diagram illustrating an approval resolution program for use in one embodiment of the present invention;

FIG. 23 is a flow diagram illustrating a change rollback process for use in one embodiment of the present invention;

FIG. 24 is a flow diagram illustrating a cancellation CGI program for use in one embodiment of the present invention.

DETAILED DESCRIPTION OF THE DRAWINGS

The present invention uses data collection and analysis tools to make the task of complex network management simpler, more efficient, and more responsive to potential system failures. Standard computer network management systems place great reliance on specialized personnel to diagnose and treat problems that develop in networks after they occur. In contrast to this reactive process of network management, the present invention streamlines data collection and utilization to allow network managers to proactively handle network configuration problems and other network management tasks. The present invention consists of a modular collection of network management tools that automates the solution of many labor- and time-intensive complex network problems.

FIG. 1 depicts a network management system 8 using the present invention and shows the functions performed by the present invention, with arrows demonstrating the flow of data among components of the present invention. The network management system 8 operates by collecting data from data sources such as a configuration change monitor 10, a multiplatform configuration file 12, a performance measurement and capacity information source 14, an application and location grouping information source 16, and/or reference table, contacts, and authorizations information 18. Each of these data sources may consist of several sources of similar data. This data is transferred through a pathway 20 to a central data store 22. The data can then be collected by a web server 24 from the data warehouse 22 via a link 26. Then, a help desk function module 28 and a configuration management function module 30 have access to the data on the web server via a link 32.

FIG. 2 is a flow diagram depicting the operation of a multipurpose data collection agent 200 which acquires the data that forms the basis of the configuration change monitors 10, multiplatform configuration files 12, and performance management and capacity information sources 14. Reference tables, contacts, and authorization information 18 are manually input via forms. In block 202, the data collection agent 200 begins executing on a computer. Then, in block 204, the data collection agent attaches to the central data store 22 and requests a work load update. In decision block 206, the data collection agent 200 determines whether there are any new instructions located in the work load table 40 in the central data store 22. If there are new instructions located in the central data store, the data collection agent 200 progresses to block 108 and updates its work load table in a local data store before moving on to query the work load table in the local data store in block 210. If no new instructions are located in the central data store, the data collection agent 200 progresses directly from decision block 206 to query the work load table in the local store in block 210.

Following the work load table query in block 210, the data collection agent 200 receives a list of items to monitor, the locations and paths of these items, the logic subroutines needed to monitor these items, and storage containers for the data it collects as shown in block 212. After receiving this information, the data collection

agent 200 determines whether any of the item locations are not local to the machine running the data collection agent at decision box 214. If any of the necessary items are not local to the executing machine, the data collection agent receives remote access subroutines and authorization information in block 216 before progressing to activate the code subroutines in block 218. If all necessary items are local to the executing machine, the data collection agent 200 proceeds directly to activate all code subroutines in block 218. The data collection agent 200 then builds an internal table, associating items to monitor with their respective locations, authorization information, and required subroutines as shown in block 220.

As depicted by iteration block 222, the data collection agent 200 proceeds through data collection steps for each item listed in the internal table built in block 220. First, the data collection agent 200 determines whether each item to monitor is local to the data collection agent's executing machine in decision box 224. For each item that is determined not to be local to the data collection agent's executing machine, the data collection agent 200 executes an access subroutine, passing location and authentication information in block 226 and then logging in to the remote machine with this authentication information at block 228. This procedure is illustrated in FIG. 3. After either logging in to the remote machine or determining that the item to monitor is local to its executing machine, the data collection agent executes the associated logic subroutine for the item while passing the item path and name of the associated storage container to the respective subroutine as shown in block 230. Then, the data collection agent 200 samples the item as defined by the associated logic subroutine in block 232 before storing the result in a specified database container as shown in block 234. The data collection agent then determines whether the monitored item was the last item to monitor as shown in decision box 236. If the monitored item was not the last item in the internal table, the data collection agent 200 returns to iteration block 222 to repeat the data collection process for the next item in the list. If the monitored item is determined to have been the last item, the data collection agent returns to block 204, once again attaching to a central data store and requesting a new work load update.

FIG. 3 is a flow diagram showing the operation of a TELNET emulation method 300 used by a data collection agent 200, the calling program, at blocks 226 and 228 to access remote computers where items to monitor are residing. The TELNET emulation method 300 receives a target network address, authentication tokens, a login prompt string, a password prompt string, and any optional secondary authentication parameters from the calling program. In the first step of the TELNET emulation method, shown in block 302, the data collection agent 200 activates and exports TELNET subroutines 400 and 500 (these TELNET subroutines, "transmit" and "waitfor" are shown in FIGS. 4 and 5, respectively). Next, in block 304, the data collection agent 200 opens a socket connection to a TELNET port at the specified address for the item to be monitored. Then, the data collection agent 200 negotiates communication session features according to the TELNET protocol specification as shown in block 306. Next, a value corresponding to the number of login attempts and denoted "loop count" is set to zero, as shown in block 308. As shown in iteration block 310, the authentication procedure for logging in to a remote system is repeated until a login is successful or until the number of login attempts equals some predetermined value, "n."

The authentication procedure begins in block 312, where the calling program transmits an end-of-line character and waits for either a login prompt string or until the number of lines returned equals some predetermined value, "m." Then, the "loop count" value is incremented by 1 as shown in box 314. Next, the access subroutine determines whether the login prompt string was successfully sent and received in decision block 316. If the login prompt string was unsuccessfully sent or received, the subroutine proceeds to decision block 318, where it is determined whether "loop count" is greater than or equal to "n." If the "loop count" is greater than or equal to "n," a failure is returned to the calling program, as shown in end-of-subroutine block 320. If the "loop count" is less than "n" at this point, the subroutine returns to iteration block 310.

If the subroutine determines the login prompt string was successfully sent and received in decision block 316, the proper authentication tokens (i.e., authentication ID) are transmitted at block 321, then "loop count" is set to zero at block 322 and the

subroutine continues to iteration block 324. From iteration block 324, the access subroutine continues until either a password is successfully sent and authorized or until "loop count" equals "n." In block 326, the access subroutine transmits an end-of-line character and waits until either a password prompt string is returned or until a number of lines equal to some predetermined value, "m," is returned. Next, in block 328, "loop count" is incremented by one. Then, in decision block 330, the subroutine determines whether "loop count" is greater than or equal to "n." If "loop count" is greater than or equal to "n," a failure is returned to the calling program at end-of-subroutine block 320. Otherwise, the subroutine transmits the authentication ID string and progresses to decision block 332 where the subroutine determines whether a password prompt or "m" lines were properly received. If the password prompt or "m" lines were not successfully received, the subroutine returns to iteration block 324. If a password prompt or "m" lines were successfully received, the subroutine proceeds to transmit the authentication password at block 333 before moving to decision block 334. In decision block 334, the subroutine determines whether optional parameters indicate additional authentication is required. If additional authentication is required, the subroutine returns to iteration block 310 to continue the authorization process. If no further authentication is required, the subroutine concludes at end-of-subroutine block 336 and returns a socket handle to the calling program.

The operation of the TELNET transmit subroutine 400 is shown as a flow diagram in FIG. 4. First, a parameter, "string" is passed to the transmit subroutine 400 at block 402. Next, the transmit subroutine 400 writes "string" to the open socket in block 404. The subroutine is then terminated at end-of-subroutine block 406, where the subroutine returns to the calling process.

FIG. 5 is a flow diagram showing the operation of a "waitfor" telnet subroutine 500. First, the parameters "string" and "numlines" are passed to the waitfor subroutine in block 502, where "string" is the value to look for and "numlines" is the number of iterations to look for "string". Next, a value "loop count" is set equal to "numlines" in block 504. The subroutine then progresses to iteration block 506, and subsequent steps iterate until "loop count" is less than or equal to zero. As shown in block 508, the subroutine reads to the end-of-line from the socket. Next, in block 510,

the subroutine decreases "loop count" by one and tests the line to see whether "string" appears in the line. In decision block 512, the subroutine determines whether "string" has been found. If "string" is found, a success is returned to the calling program at end-of-subroutine block 514. If "string" is not found, the subroutine continues to decision block 516 to determine whether "loop count" is less than or equal to zero. If "loop count" is less than or equal to zero, the waitfor subroutine 500 returns a failure to the calling program at end-of-subroutine block 518. If "loop count" is greater than zero, the waitfor subroutine returns to iteration block 506 to begin once again testing for "string" at the socket.

FIG. 6 is a flow diagram illustrating a configuration change monitor subroutine 600 for use by multipurpose data collection agent 200. The configuration change monitor subroutine 600 is given parameters including file paths and storage containers for checksums and files. First, in block 602, the configuration change monitor subroutine 600 opens the file specified for the item to be monitored. Then, as shown in block 604, the configuration change monitor subroutine 600 reads data from the corresponding file. In block 606, the subroutine calculates a checksum for the data read in block 604. The configuration change monitor subroutine then resumes to block 608, where the subroutine looks up the last known checksum for the item to be monitored in a local data store. Next, at decision block 610, the subroutine determines whether the calculated checksum from block 606 and the last known checksum from block 608 are different. If the two checksums are not different, the subroutine returns to the calling process (i.e., the multipurpose data collection agent) at the end-of-subroutine block 620. If the two checksums are different, the new checksum is stored as the last known checksum for the item in the local data store as shown in block 612. Following this, the subroutine attaches to the central data store 22 in block 614. Next, the new checksum is stored as the last known checksum for this item and is timestamped as shown in block 616. Next as shown in block 617, a new record is inserted into the configuration changes table 42 in the central data store 22. This new record contains the item ID and a timestamp. The subroutine then stores the changed configuration file as read in block 604 in a central store with a

timestamp, as shown in block 618. The subroutine then returns to the calling process in end-of-subroutine block 620.

FIG. 7 is a flow diagram for a web server performance monitor subroutine 700 called by multipurpose data collection agent 200 when data must be collected on web server performance. The web server performance monitor subroutine 700 is first given an ordered list of uniform resource locators ("URLs") to test, a pre-opened database connection handle, and storage containers for performance measurements and errors.

The web server performance monitor subroutine 700 begins in block 702 by opening the initial URL in the ordered list of URLs to test. At this point, the web server performance monitor subroutine 700 tests the return code from the URL. Next, in decision block 704, the web server performance monitor subroutine determines whether the return code indicates a redirection. If a redirection is indicated, the subroutine 700 proceeds to increment the page count, accumulate the bytes received from the URL, accumulate the time to download the page, open the specified URL and check the return code from the new URL as shown in block 706. From block 706, the subroutine 700 returns to decision block 704 to determine whether the new return code indicates redirection. If the new return code does not indicate redirection, the subroutine 700 progresses to decision block 708. If redirection continues to be indicated, though, the subroutine 700 continues to loop through blocks 706 and 704 until no redirection is indicated. At decision block 708, the subroutine 700 determines whether the return code from the URL indicates failure.

If a failure is indicated, an error code is stored with a timestamp in the specified storage container 44 and error table 46 (see FIG. 1a) at block 710. Under this "failure indicated" pathway, the subroutine 700 then proceeds to determine whether the failure-returning URL is the last URL in the list at decision block 712. If the URL is the last in the list, the subroutine proceeds to end-of-subroutine block and returns to the calling process (i.e., the multipurpose data collection agent 200). If the URL is not the last in the URL list, the subroutine 700 returns to block 702 and opens the next URL in the list.

If no failure is indicated at decision block 708, the subroutine 700 increments the page count, and accumulates the number of downloaded bytes and download time for the non-failure-returning URL at block 716. At this step, the subroutine 700 parses the URL for referenced files and progresses to iteration block 718 which repeats a performance monitoring loop for each referenced file. The subroutine 700 moves to decision block 720 where it determines whether the referenced filename exists in a cache on the machine running the subroutine 700. If the filename exists in the cache, the subroutine returns to iteration block 718 and proceeds to the next referenced file. If the filename does not exist in the cache as determined at decision block 720, the subroutine 700 resumes to download the new file, increment the page count, and record the byte count and download time at block 722. The filename is also recorded in a cache at this block. Next, the subroutine 700 progresses to decision block 724 to determine whether the previously tested file is the last referenced file at the listed URL. If the tested file is not the last referenced file, the subroutine 700 returns to iteration block 718 and moves on to the next referenced file. If the tested file is the last referenced file, the subroutine 700 proceeds to block 726, where it stores the URL, the page count, the number of downloaded bytes, the download time, and an associated timestamp in a performance table 48 located in the central data warehouse 22 (see FIG. 1a). Next, the subroutine 700 determines whether the tested URL is the last URL in the list of URLs to test at decision block 712. If the tested URL is not the last URL, the subroutine 700 returns to block 702 and if the tested URL is the last URL, the subroutine ends at end-of-subroutine block 714 and returns to the calling process (i.e., the multipurpose data collection agent 200).

A router capacity and performance monitor subroutine 800, another logic subroutine that can be called by the multipurpose data collection agent 200 is shown as a flow diagram in FIG. 8. The router capacity and performance monitor subroutine 800 depends on the TELNET emulation method 300 to access information at routers. The router capacity and performance monitor subroutine 800 is provided with a pre-opened, pre-authenticated socket handle, a command string, a prompt string, a logoff string, a parsing subroutine, a pre-opened database handle, a router ID, and storage containers for data and errors by the multipurpose data collection agent 200.

The router capacity and performance monitor subroutine 800 begins by transmitting an end-of-line character to the router to be monitored at block 802. Next, the monitor subroutine 800 waits for a prompt string from the router to be monitored in block 804. At decision block 806, the monitor subroutine 800 determines whether the prompt string has been successfully returned. If there is no successful prompt string returned, the monitor subroutine 800 logs the router ID, the error, and a timestamp to the specified error container provided by the multipurpose data collection agent. Following this "prompt string failure" pathway, the monitor subroutine 800 moves on to close the pre-opened and pre-authenticated socket handle at block 810 before returning to the multipurpose data collection agent 200 at end-of-subroutine block 812.

If a successful prompt string return is found at the prompt string decision block 806, the monitor subroutine 800 transmits the command string provided by the multipurpose data collection agent 200 to the router to be tested at block 814. The monitor subroutine 800 then waits for a prompt string from the router as shown in block 816, and captures the output from the router to a variable residing in volatile memory. Then, if the output capture is determined not to be successful at decision block 818, the monitor subroutine 800 proceeds to block 808, recording the router ID, error, and a timestamp to the specified error container as described above before proceeding to blocks 810 and 812.

If the output capture at block 816 is deemed successful by the monitor subroutine 800 at the output capture decision block 818, the monitor subroutine 800 passes the captured data, router ID and storage container names to a specified parsing subroutine at block 820. The specified parsing subroutine begins in block 822. First, the parsing subroutine validates the data format from the router at block 824. The parsing subroutine determines whether the format is acceptable at decision block 826. If the format is unacceptable, the parsing subroutine terminates and the monitor subroutine 800 logs the router ID and error with an associated timestamp to a specified error container at block 808 before terminating as described above. If the parsing subroutine determines the router data format to be acceptable at the format decision block 826, the parsing subroutine progresses to iteration block 828 and

begins a parsing loop for each interface at the router. For each interface, the parsing subroutine parses data from the monitored router into variables and constructs an insert statement containing interface statistics at block 830. The parsing subroutine then stores a timestamp, the monitored router ID, and data points for the monitored router in the central data store 22 (i.e., the "main store") as shown in block 832 and moves on to determine whether it has tested the last router interface at decision block 834. If the monitored interface is not the last interface, the parsing subroutine returns to iteration block 828 and moves on to the next interface. If the monitored interface is the last interface at the router, the parsing subroutine transmits a logoff string as shown in block 836 and returns to the monitor subroutine at block 810 The specified socket handle is then closed and the monitor subroutine terminates at end-of-subroutine block 812, returning to the calling program (i.e., the multipurpose data collection agent 200).

FIG. 9 is a flow diagram of a router configuration change monitor subroutine 900, another subroutine that may be called by the multipurpose data collection agent 200 to monitor routers. The router configuration change monitor subroutine 900 relies on the TELNET emulation method 300 to access information at routers. The router configuration change monitor subroutine 900 is provided with a pre-opened, pre-authenticated socket handle, a command string, a prompt string, a logoff string, a pre-opened database handle, a router ID, and storage containers for data and errors by the multipurpose data collection agent 200 prior to its execution.

After being provided with these parameters, the router configuration change monitor subroutine 900 transmits an end-of-line character to the router to be monitored, as shown in block 902. The router configuration change monitor subroutine 900 then waits for a prompt string from the router to be monitored at block 904. If the prompt string is deemed not to have been successfully sent and received at decision block 906, the router configuration monitor subroutine 900 proceeds to log the router ID, an error message, and a corresponding timestamp to a specified error container at block 908. Following this "prompt string failure" pathway, the router configuration monitor subroutine 900 proceeds to close the pre-opened, pre-

authenticated socket handle at block 910 before returning to the calling program (i.e., the multipurpose data collection agent) at end-of-subroutine block 912.

If the router configuration monitor subroutine 900 determines the prompt string to have been successfully transmitted and received at decision block 906, the router configuration monitor subroutine 900 resumes to transmit a command string at block 914. The router configuration monitor subroutine 900 then waits for a prompt string and captures the router output to a variable, as shown in block 916. If the output is not successfully captured, the router configuration monitor subroutine 900 then progresses from decision block 918 to log the router ID, an error message, and an associated timestamp to the specified error container at block 908 before proceeding through block 910 and terminating at block 912 as described above.

If the output is deemed to be successfully captured at decision block 918, the router configuration monitor subroutine 900 calculates a checksum for the captured data in block 920. The router configuration monitor subroutine 900 then retrieves the last known checksum for the monitored router configuration in block 922. Next, if the checksums are found not to be different at decision block 924, the router configuration monitor subroutine 900 determines that the router configuration has not changed and transmits the logoff string, as shown in block 926, before passing through blocks 910 and 912 and terminating as described above.

If the checksums are determined to be different at decision block 924, the router configuration monitor subroutine 900 must update the data in the main store to reflect a configuration change. First, the router configuration monitor subroutine 900 stores the new checksum as the last known checksum in a local data store in block 928. The router configuration monitor subroutine 900 then resumes to open a connection to the main data store, 22, at block 930. Next, the router configuration monitor subroutine 900 stores a timestamp, router ID, and new checksum as the last known checksum in the main data store 22 at block 932. After storing this data, the router configuration monitor subroutine 900 transmits a logoff string as shown in block 926 and closes the socket handle in block 910 before terminating and returning to the multipurpose data collection agent 200.

FIG. 10 is a flow diagram of a generic capacity monitor subroutine 1000, another subroutine which can be called by the multipurpose data collection agent 200 to monitor remote computers. The capacity monitor subroutine 1000 is provided with a command path, a parsing subroutine, a pre-opened database handle, and a storage container for capacity data by the multipurpose data collection agent 200 before execution.

When the capacity monitor subroutine 1000 begins, it opens a piped output from the specified command path as shown in block 1002. Next, the capacity monitor subroutine 1000 determines whether the pipe was successfully opened at decision block 1004. If the pipe is not successfully opened, the capacity monitor subroutine 1000 proceeds to store an error to the specified error container in the central data store as shown in block 1006 before returning to the calling process (i.e., the multipurpose data collection agent) as shown in end-of-subroutine block 1008. If the pipe is determined to be successfully opened at decision block 1004, the capacity monitor subroutine 1000 proceeds to block 1010, where it reads all data from the pipe into a memory variable. Next, the capacity monitor subroutine 1000 determines whether it properly receives the data at decision block 1012. If the capacity monitor subroutine 1000 does not properly receive the data, it stores a timestamp and error message to the specified error container as shown in block 1006 before terminating as described above at end-of-subroutine block 1008.

If the capacity monitor subroutine 1000 determines that it correctly received the data at decision block 1012, the capacity monitor subroutine 1000 proceeds to pass a data variable, and storage container IDs to its parsing subroutine as specified by the multipurpose data collection agent 200. Next, the specified parsing subroutine is started as shown in block 1016.

The parsing subroutine first validates that the data received is in the correct format as shown in blocks 1018 and 1020. If the data is not in the correct format, the parsing subroutine terminates and the capacity monitor subroutine 1000 stores a timestamp and error message to the specified error container in block 1006 before terminating at end-of-subroutine block 1008 as described above.

If the parsing subroutine determines that the data is in the correct format at decision block 1020, the parsing subroutine proceeds to extract data points into corresponding variables at block 1022. Next, the parsing subroutine constructs database insert statements at block 1024 before storing the capacity data to a specified container as shown in block 1026. After storing the capacity data to the correct container, the parsing subroutine and capacity monitor subroutine 1000 terminate and return to the multipurpose data collection agent 200 at end-of-subroutine block 1008.

Another subroutine that can be called by the multipurpose data collection agent 200 is the customer database availability and performance monitor subroutine 1100, which is shown as a flow diagram in FIG. 11. The customer database availability and performance monitor subroutine 1100 depends on external software for its operation. This external software includes software for network connectivity, a software developer's kit for the target database management system, and a pre-catalogued data source. The multipurpose data collection agent 200 must pass on other information to the customer database availability and performance monitor subroutine 1100 for it to function properly. This passed information includes a target data source name, authentication tokens, a pre-defined test query, a query ID, a parsing subroutine name, a pre-opened database handle for the central data store 22, and storage containers for performance data and errors.

The customer database availability and performance monitor subroutine 1100 begins its operation by opening a target database connection using the authentication tokens provided by the multipurpose data collection agent, as shown in block 1102. Next, the customer database availability and performance monitor subroutine 1100 determines whether the connection succeeds at decision block 1104. If the connection does not succeed, the customer database availability and performance monitor subroutine 1100 proceeds to store the corresponding query ID, an error message, and a timestamp to the specified error container at block 1106 before terminating and returning to the calling process (i.e., the multipurpose data collection agent) at end-of-subroutine block 1108.

If the connection is determined to succeed at decision block 1104, the customer database availability and performance monitor subroutine 1100 starts a timer

as shown in block 1110. The customer database availability and performance monitor subroutine 1100 then issues a test query and stores the set of results in a local variable in block 1112. Next, the customer database availability and performance monitor subroutine 1100 ends the timer begun in block 1110, calculates the elapsed time, and stores the elapsed time in a local variable as shown in block 1114. The customer database availability and performance monitor subroutine 1100 then determines whether the query succeeded at block 1116. If the query did not succeed, the customer database availability and performance monitor subroutine 1100 moves on to store the query ID, an error message, and a corresponding timestamp to an error container as shown in block 1106 before terminating at end-of-subroutine block 1108 as described above.

If the query did succeed at block 1116, the customer database availability and performance monitor subroutine 1100 must begin the parsing subroutine named by the multipurpose data collection agent 200. First, the customer database availability and performance monitor subroutine 1100 passes the local variable from query results, corresponding storage container names, and the local elapsed time variable to the specified parsing subroutine as shown in block 1118. Next, the specified parsing subroutine is begun at block 1120. The parsing subroutine first determines whether the data provided by the customer database availability and performance monitor subroutine 1100 is in the correct format as shown in block 1122 and decision block 1124. If the data is not in the correct format, the parsing subroutine terminates and the query ID, an error message, and a corresponding timestamp are stored to the specified error container at block 1106. The customer database availability and performance monitor subroutine 1100 then terminates as described above at end-of-subroutine block 1108.

If the data format is correct at decision block 1124, the parsing subroutine counts the number of rows of data returned and stores the result as a local variable at block 1126. Next, the parsing subroutine stores the query ID, the variable corresponding to the number of rows (i.e., "rowcount"), the elapsed time determined at block 1114, and a corresponding timestamp in the specified storage container at

block 1128. The customer database availability and performance monitor subroutine 1100 then terminates as described above at end-of-subroutine block 1108.

The network management system 8 utilizes the information gathered by the multipurpose data collection agent and its called subroutines to provide a wide variety of possible network management functions. One critical function provided by the network management system 8 is a help desk function, which is carried out by the help desk function module 28. The help desk function module 28 uses a combination of human expertise and data provided by the data collection procedures described above to deliver responsive solutions quickly and efficiently. Failure resolution is handled through a ticket-based system, and a ticket eventually contains such information as the source of the failure, candidate causes for the failure, and the ideal contact to resolve the failure. The help desk function 28 can be divided into four separate phases: the ticket creation phase 1300, the ticket assignment phase 1400, the ticket resolution phase 1500, and the resolution or rejection common gateway interface ("CGI") application 1600. A logical overview 1200 of the help desk function module is shown in FIG. 12.

A problem resolution process within the help desk function module begins at block 1202 when a ticket is created in the ticket creation phase 1300. Next, as shown in block 1204, the ticket is assigned to an identified assignee capable of solving the problem in the ticket assignment phase 1400. The assignee then processes the ticket in the ticket resolution phase 1600, as shown in block 1206. The assignee then uses a resolution or rejection CGI application 1700 to report the resolution of the problem or the rejection of the ticket at block 1208. As shown in decision block 1210, if the ticket is rejected, the ticket is reassigned in the ticket assignment phase 1400. If the ticket is not rejected at decision block 1210, the ticket and its resolution are added to a history table 50 in the central data store 22 and the ticket is removed from a processing table 52 in the central data store 22, as shown in block 1212. This terminates a problem resolution process as shown in end-of-process block 1214.

The help desk function's ticket creation phase 1300 is shown as a flow diagram in FIG. 13. The ticket creation phase consists of a ticket creation subroutine which may be called either by helpdesk personnel responding to a user-reported

failure or by an automated monitor automatically discovering a failure, as shown in block 1306. If a user reports a failure to a computer helpdesk as shown in block 1302, helpdesk personnel can then manually enter the failed application, its location, the reported failure time, and the last known working time into the ticket creation subroutine as shown in block 1304. The ticket creation subroutine then executes as shown in block 1308. The ticket creation subroutine then queries the central data store 22 to determine whether it contains last time the failed application was functioning successfully as shown in decision block 1310. If the ticket creation subroutine discovers the last successful time is not known, the ticket creation subroutine proceeds to look up the last successful test time in block 1312. The ticket creation subroutine then creates an insert statement which contains the application, the location, the failure time, and the last successful time as recorded in the ticket creation subroutine's memory variables, as shown in block 1314. If the last successful time is known at decision block 1310, the ticket creation subroutine proceeds directly to creating the insert statement at block 1312 without looking up or inserting the last successful time into the insert statement. Next, the ticket creation subroutine stores all of the information in the insert statement in a reported problems table 54 in the central data store 22. Finally, the ticket creation subroutine terminates at the end-of-subroutine block 1318 and returns to the calling program, which may be an automated availability monitor or a help desk manual entry form.

After creating the ticket, the help desk function module 28 must assign the ticket to the correct person to resolve the failure, the assignee. This is handled in the ticket assignment phase 1400, which is shown as a flow diagram in FIG. 14. First, as shown in block 1402, the help desk function module 28 gets a list of active tickets from a reported problems table 54 located in the central data store 22. Next, in block 1404 the help desk function module 28 accesses the central data store 22 to get a group of ticket IDs, the tickets' statuses, the tickets' assignees, and the tickets' assigned causes. The help desk function module 28 then progresses to iteration block 1406, passing through a ticket assignment routine for each ticket. First, the help desk function module 28 determines whether the current ticket is pending at decision block 1408. If the ticket is pending, it is currently assigned, and the help desk function

module 28 passes through block 1410 to iteration block 1406 to address the next ticket. If the ticket is not pending, the help desk function module 28 determines whether the ticket has been rejected at decision block 1412. A ticket is rejected by the ticket's assignee when, for example, the assignee determines that the candidate cause identified on the ticket is incorrect. If the ticket has been rejected, the help desk function module 28 deletes the previously identified candidate cause from the problem candidates table 56 specified by the assigned candidate, as shown in block 1414. Following the "rejected status" pathway, the help desk function module 28 then proceeds to execute a candidate cause identification subroutine 1500 (shown in FIG. 15) to look up relevant information in the central data store 22, as shown in block 1415. If the ticket has not been rejected, the help desk function module 28 passes directly to block 1416 from block 1412.

Next, the help desk function module 28 updates the assigned cause and assignee in a reported problems table 54 in the central data store 22, as shown in block 1418. The help desk function module 28 then proceeds to create a ticket information web page in block 1420 and send a notification to the identified assignee in block 1422. Next, if the help desk function module 28 determines that it has processed the last ticket at the "last ticket" decision block 1424, it puts the ticket assignment phase into a "sleep mode" in block 1426 until the next ticket assignment cycle is initiated. If the help desk function module 28 has not assigned the last ticket on its list of active tickets, it proceeds to block 1410 where the process of assigning the next ticket begins.

FIG. 15 is a flow diagram showing the candidate cause identification subroutine 1500 called during the ticket assignment phase at block 1415, as shown in FIG. 14. The candidate cause identification subroutine 1500 is first given necessary information such as a reporting network address, the failed application, a first timestamp for the last known successful operation of the problem item, and a second timestamp noting when the problem was reported. As shown in block 1502, the candidate cause identification subroutine 1500 first looks up a service area value from a network map table 58 residing in the central data store 22 (see FIG. 1a). Next, as shown in block 1504, the candidate cause identification subroutine 1500 creates a

temporary table, "A," which it populates with a list of configuration item IDs from a location and application table 60 in the central data store 22. This list identifies configuration item IDs for all system components associated with the specified application and service area identified in the previous step. The candidate cause identification subroutine 1500 then creates a second temporary table, "B," which it populates with a list of configuration item IDs with changed checksums during the specified timeframe from the configuration change table 42 located in the central data store 22 in block 1506.

The candidate cause identification subroutine 1500 next retrieves a list of configuration item IDs common to both temporary tables it has created, as shown in block 1508. Next, at block 1510, the candidate cause identification subroutine 1500 looks up contact names and email addresses for all configuration item IDs in the list created at block 1508. This information is retrieved from a technical and approval contacts reference table 62 in the central data store 22 (see FIG. 1a). The candidate cause identification subroutine 1500 terminates when it returns the values it has joined to the calling program at end-of-subroutine block 1510.

Once tickets are assigned by the help desk function module 28, the next step in addressing a failure is to resolve the ticket. This is done in the ticket resolution phase 1600, shown as a flow diagram in FIG. 16. The ticket resolution phase begins when an assignee, identified at the ticket assignment phase, receives notification of a ticket in block 1602. After receiving notice of a ticket, the assignee reads the ticket information web page created in the ticket assignment phase, as shown in block 1604. Next, the assignee attempts to resolve the problem identified on the ticket information web page, as shown in block 1606. If the assignee does not find a problem at decision block 1608, the assignee fills out a rejection form as shown in block 1610 and the rejection form is submitted by the help desk function module 28 to the rejection common gateway interface application 1700 (shown in FIG. 17 and described below) before the ticket resolution phase is ended at end-of-procedure block 1614.

If the assignee finds a problem at decision block 1608, the assignee takes corrective action as shown in block 1616 and then fills out a resolution form as shown in block 1618. The help desk function module 28 then progresses to block 1620 and

submits the resolution form to the resolution common gateway interface application 1700, terminating the ticket resolution phase at the end-of-procedure block 1614.

The resolution or rejection of tickets is managed by a resolution or rejection common gateway interface application 1700. In a common gateway interface ("CGI") application, data is usually passed back and forth between a user and a web server. The CGI application 1700 receives rejection or resolution forms from the ticket resolution phase 1600 and then updates relevant records in the central data store 22 to reflect the resolution or rejection of help desk tickets. The CGI application 1700 first parses form data from either a rejection form or a resolution form in block 1702. Next, the CGI application 1700 determines whether the ticket was rejected at decision block 1704. If the ticket was rejected, the CGI application 1700 creates a structured query language ("SQL") statement from the parsed form data in block 1706. An SQL statement is a standard method of receiving information from or updating a database, and here it is sent to the central data store 22. Next, a status field in a reported problems table 54 in the central data store 22 is updated to "rejected" at block 1708 before the CGI application 1700 terminates at end-of-application block 1710.

If the CGI application 1700 determines that a ticket was not rejected at decision block 1704, the CGI application 1700 proceeds to block 1712 and looks up the original ticket information from the reported problems table 54 in the central data store 22. Next, the CGI application 1700 creates an SQL statement from the resolution form data and the original ticket information at the central data store 22, as shown in block 1714. The CGI application 1700 then proceeds to block 1716 where it records the closed ticket information in a problem history table 64 located in the central data store 22. Then, at block 1718 the CGI application 1700 deletes all records for the ticket ID from a reported problems table 54 and a problem candidates table 56 located in the central data store 22 before terminating at the end-of-application block 1710.

Another function provided by the network management system 8 is a configuration management function, which is handled by configuration management function module 30. Configuration changes are the basis of a large number of computer network failures, and thus the ability to manage configuration changes is a

key tool in efficient network management. When a configuration change is detected by, for example, the configuration change monitor 600 or the router configuration change monitor 900, the configuration management function module 30 must determine how to resolve the configuration change. This module uses technical and approval contacts to determine whether configuration changes should be approved. Like the help desk function module 28, the configuration management function module 30 relies on a ticket-based system to determine whether configuration changes should be approved.

A logic overview of the configuration management function module 30 is shown as a flow diagram in FIG. 18. First, an approval ticket is created and sent to approval contacts during the approval request phase 1900 (shown in FIG. 19), as shown in block 1802. Next, in the approval resolution phase 2000 (shown in FIG. 20), the configuration management function module 30 checks responses from approval contacts. The configuration management function module 30 then determines whether all contacts have approved a configuration change as shown in decision block 1806. If any contacts have not approved the change, then as shown in block 1808 the original configuration is restored in the change rollback process 2300 (shown in FIG. 23) before the current configuration management process ends at end-of-process block 1810. If all contacts are determined to have approved the configuration change at decision block 1806, the configuration management function module authorizes the changed configuration file as shown in block 1812. Then, the current configuration management process ends at end-of-process block 1810.

The configuration management function module 30 first addresses configuration changes by initiating the approval request phase 1900, shown as a flow diagram in FIG. 19. In the approval request phase 1900, the configuration management function module 30 requests approval from approval and technical contacts (i.e., "approvers") for identified configuration changes and notifies all contacts of proposed configuration changes. First, a database trigger detects an insert from, for example, the configuration change monitor subroutine 600 or the router configuration change monitor subroutine 900 into a configuration change table 42 in the central data store 22, as shown in block 1902. Next, the configuration

management function module 30 begins an approval request application at block 1904. The approval request application gets the item ID, description, and technical and approval contacts from reference tables 62 located in the central data store 22, as shown in block 1906. The approval request application then retrieves copies of approved and "last seen" configuration files for each item at block 1908 before creating an approval ticket ID at block 1910. After the approval ticket ID is created, the approval request application creates a web page ticket with three frames: one frame each for the old and new configurations and a third frame containing an approval request form.

Next, as shown in iteration block 1914, the approval request application follows a notification routine for each contact identified in block 1906. The approval request application constructs an SQL insert statement with a ticket ID and contact ID, and assigns the ticket a status of "pending" at block 1916. The approval request application then inserts a record into a pending approvals table 66 in the central data store 22 at block 1918. Next, the approval request application sends an email notification to the contact, referencing the web page ticket created in block 1912 and requesting approval. The approval request application then determines whether it has sent an email to each contact found in block 1906 at decision block 1922. If the approval request application has not sent an email to each contact, it returns to iteration block 1914. If the approval request application has sent an email to each contact, it terminates at end-of-phase block 1924.

The next step in resolving configuration changes is the contact approval phase 2000, which is shown as a flow diagram in FIG. 20. In the contact approval phase, the approver receives email notification of an approval request with a reference to an approval web page, as shown in block 2002. This email was sent in block 1920 as shown in FIG. 19. The approver then accesses the referenced web page and reviews the old and new configurations as shown in block 2004. The approver next fills out the approval form on the referenced web page and submits the approval form to the approval CGI program 2100, shown in FIG. 21. This terminates the contact approval phase as shown in the end-of-phase block 2008.

The approval form submitted in block 2006 is sent to an approval CGI program 2100, which is shown as a flow diagram in FIG. 21. The approval CGI program first parses the data from the approval form in block 2102. Next, the approval CGI program 2100 determines whether the proposed configuration change has been approved at decision block 2104. If the configuration change has not been approved by the ticket's contact, the approval CGI program 2100 creates an SQL update with a status of "denied" for the approval ticket ID and contact ID associated with the approval form, as shown in block 2106, and then updates the record in the pending approvals table 66 in the central data store 22 as shown in block 2108 before terminating at end-of-program block 2112. If the configuration change has been approved by the ticket's contact, the approval CGI program 2100 creates an SQL update with a status of "approved" for the approval ticket ID and contact ID associated with the approval form, then updates the record in the pending approvals table 66 in the central data store before terminating at end-of-program block 2112.

The next component of the configuration management function module 30 is an approval resolution program 2200 which tracks proposed changes based on the status of records of configuration changes and change attempts in the central data store 22. The approval resolution program 2200 first gets a list of all configuration change approval tickets from a pending approvals table 66 in the central data store 22, as shown in block 2202. Then, the approval resolution program 2200 moves on to iteration block 2204 and for each approval ticket ID received in block 2202 the approval resolution program 2200 gets status records 68 from the central data store 22. Next, at decision block 2208, the approval resolution program 2200 determines whether any records corresponding to each ticket ID are pending. If any records have a "pending" status, the approval resolution program 2200 proceeds through block 2210 to address the next approval ticket at iteration block 2204.

If no records have a "pending" status, the approval resolution program 2200 moves on to decision block 2212 where it determines whether any records have been given a "cancel" status. A cancel status is assigned by a technical contact after a denied configuration change has been backed out by the technical contact. If a "cancel" status has been found, the change no longer exists and all records for the

cancelled ticket are deleted from the central data store 22 at block 2214 before the approval resolution program 2200 addresses the next ticket.

If no records have a "cancel" status at decision block 2212, the approval resolution program 2200 proceeds to decision block 2216 where it determines whether any records have a "denied" status. If any records have a "denied" status, the approval resolution program 2200 proceeds to block 2218 where it looks up the item description, the old configuration for that item, the new configuration for that item, and the technical contact for the changed item in the central data store 22. All records for the ticket are then updated to "pending" at block 2220. Next, the approval resolution program 2200 generates a "rollback ticket" web page with three frames: one for each of the old and new configurations, and one containing instructions to back out of the change, along with a confirmation form at block 2222. The approval resolution program 2200 then moves to block 2224, sending an email notification to the proper technical contact directing the technical contact to the "rollback ticket" web page. At the end of this "ticket denied" pathway, the approval resolution program 2200 terminates at end-of-program block 2226.

If no records are found with a "denied" status at decision block 2216, the approval resolution program 2200 then double-checks at decision block 2228 whether all records have received an "approved" status from a technical contact. If each record is not approved, the approval resolution program 2200 sends an email notification to a system administrator that something is wrong with the configuration change approval system, as shown in block 2230. If all records have an "approved" status, the approval resolution program 2200 looks up the ID and checksum for the new configuration file in the central data store 22 as shown in block 2232. As shown in block 2233, the approval resolution program 2200 then applies predetermined rules to the approved configuration changes. For example, at this point the approval resolution program 2200 might change the configurations of all similarly situated items to the newly approved configuration. Next, the approval resolution program 2200 updates the authorized checksum for the identified item in the central data store 22 with a new checksum as shown in block 2234. Finally, the approval resolution program 2200 terminates at end-of-program block 2226.

If a proposed configuration change is unacceptable, the configuration management function module 30 initiates the change rollback process 2300 which is shown as a flow diagram in FIG. 23. The change rollback process 2300 begins at block 2302 when a technical contact reads the email notification sent by the approval resolution program in block 2224 (shown in FIG. 22) and accesses the rollback ticket web page generated in block 2222. The technical contact next manually restores the configuration file as shown in block 2304. Next, at block 2306, the technical contact checks a confirmation box on the confirmation form and submits the form to a cancellation CGI program 2400, shown in FIG. 24. This terminates the change rollback process 2300 as shown in end-of-process block 2308.

The next step performed in the configuration management function module 30 is carried out by a cancellation CGI program 2400, shown as a flow diagram in FIG. 24. As seen in block 2402, the cancellation CGI program 2400 first parses the data in the form submitted in block 2306, shown in FIG. 23. Next, as shown in block 2404, the cancellation CGI program 2400 updates all records in the pending approvals table 66 in the central data store 22 for the ticket ID corresponding to the confirmation form it received, giving the records a status of "cancel." The cancellation CGI program then terminates at end-of-program block 2406.

While the present invention has been described with reference to one or more preferred embodiments, those skilled in the art will recognize that many changes may be made thereto without departing from the spirit and scope of the present invention which is set forth in the following claims.

WHAT IS CLAIMED IS:

1. A method of managing computer networks containing a server and multiple client computers, comprising:

collecting and storing in a central data store configuration change information from data collection agents executing on said client computers;

collecting and storing in a central data store multi-platform configuration files from said data collection agents;

collecting and storing in a central data store performance measurement and capacity information from said data collection agents;

collecting and storing in a central data store application and location grouping information from said data collection agents;

collecting and storing in a central data store reference table information from said data collection agents; and

providing said configuration change information, multi-platform configuration files, performance measurement and capacity information, application and location grouping information, and reference table information to a help desk function module and a configuration management function module.

2. The method of claim 1, wherein the information in said central data store is updated when a configuration change or capacity alert is detected and each update of said operations data warehouse is timestamped.

3. The method of claim 1, wherein information in said central data store is grouped into application groups and location groups.

4. The method of claim 4, wherein said help desk application responds to help desk requests from a requester by:

accessing a corresponding application or location group to identify candidate causes for a related malfunction;

identifying candidate causes corresponding to the related application, the location of said requester, and the time of the request;

identifying a related contact; and

sending a work order to said contact specifying the candidate causes.

5. A method of monitoring configuration changes in a network comprising:

executing configuration data collection agents on one or more computers, said data collection agents tracking and time-stamping configuration changes to said computers; and

reporting said configuration changes to a central data store.

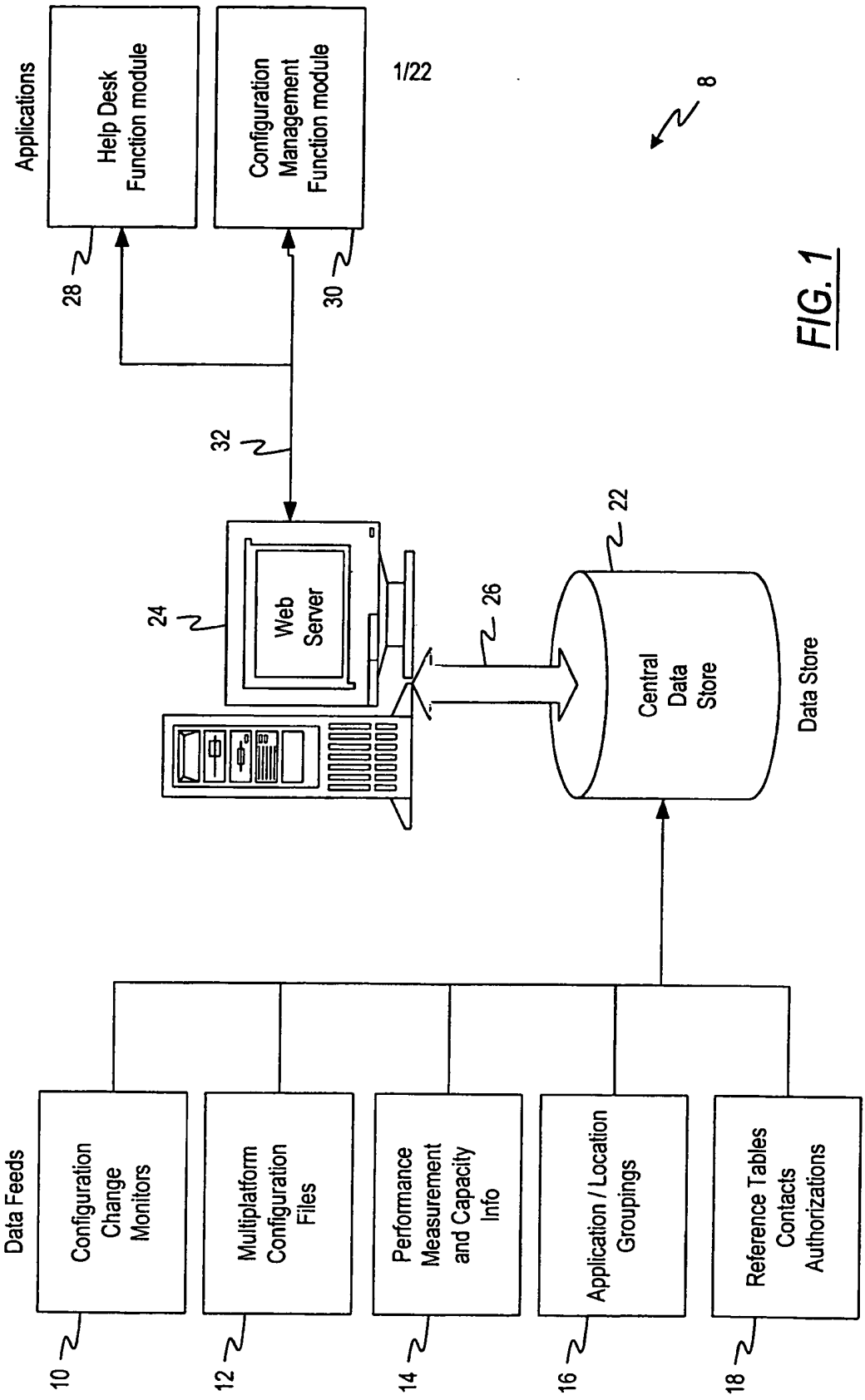

Applications
Help Desk Function module
Configuration Management Function module
1/22
8
28
30
32
24
Web Server
26
22
Central Data Store
Data Store
Data Feeds
Configuration Change Monitors
Multiplatform Configuration Files
Performance Measurement and Capacity Info
Application / Location Groupings
Reference Tables Contacts Authorizations
10
12
14
16
18

FIG. 1

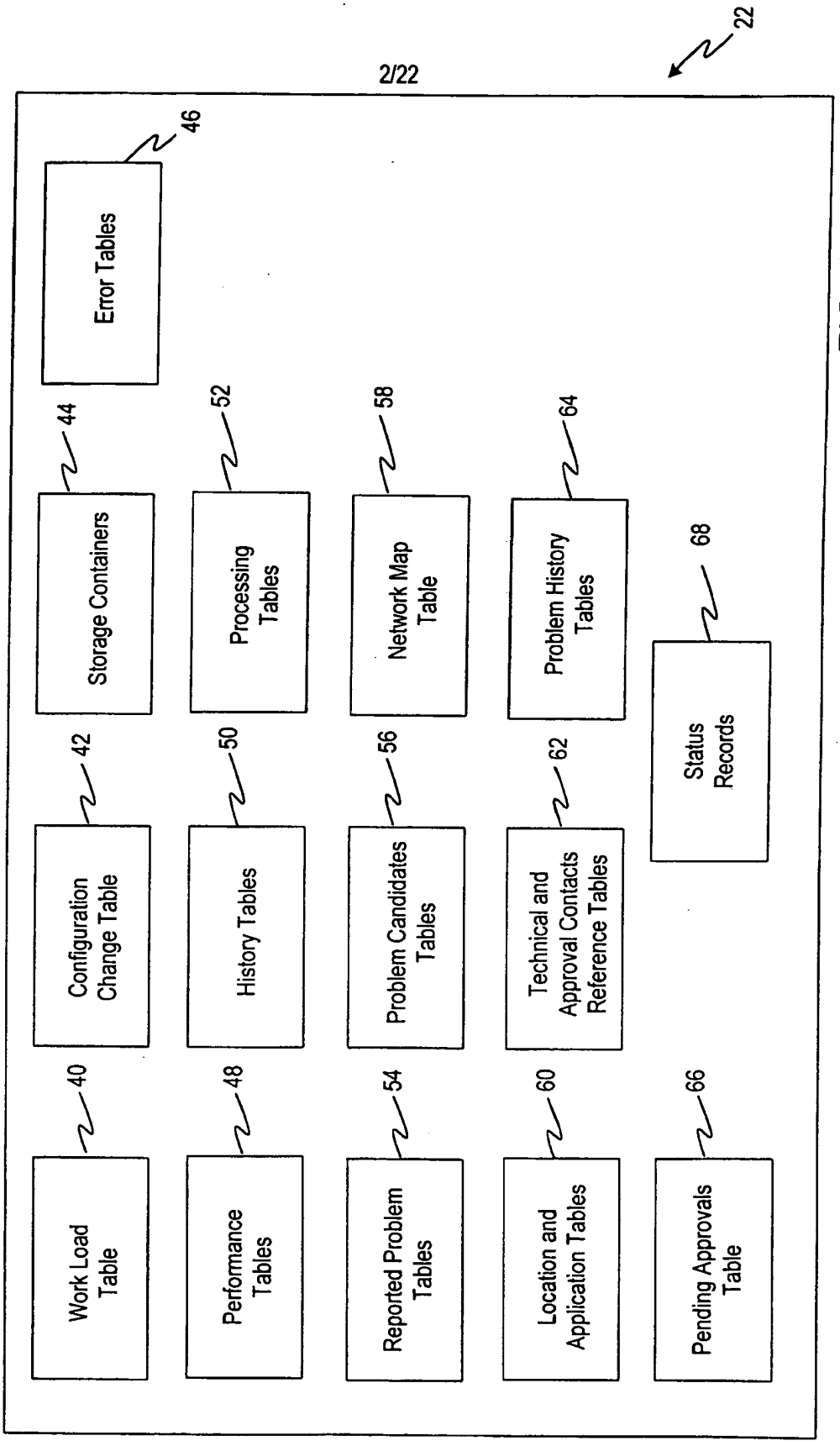

22
Error Tables
46
Storage Containers
44
Processing Tables
52
Network Map Table
58
Problem History Tables
64
Status Records
68
Configuration Change Table
42
History Tables
50
Problem Candidates Tables
56
Technical and Approval Contacts Reference Tables
62
Work Load Table
40
Performance Tables
48
Reported Problem Tables
54
Location and Application Tables
60
Pending Approvals Table
66

FIG. 1a

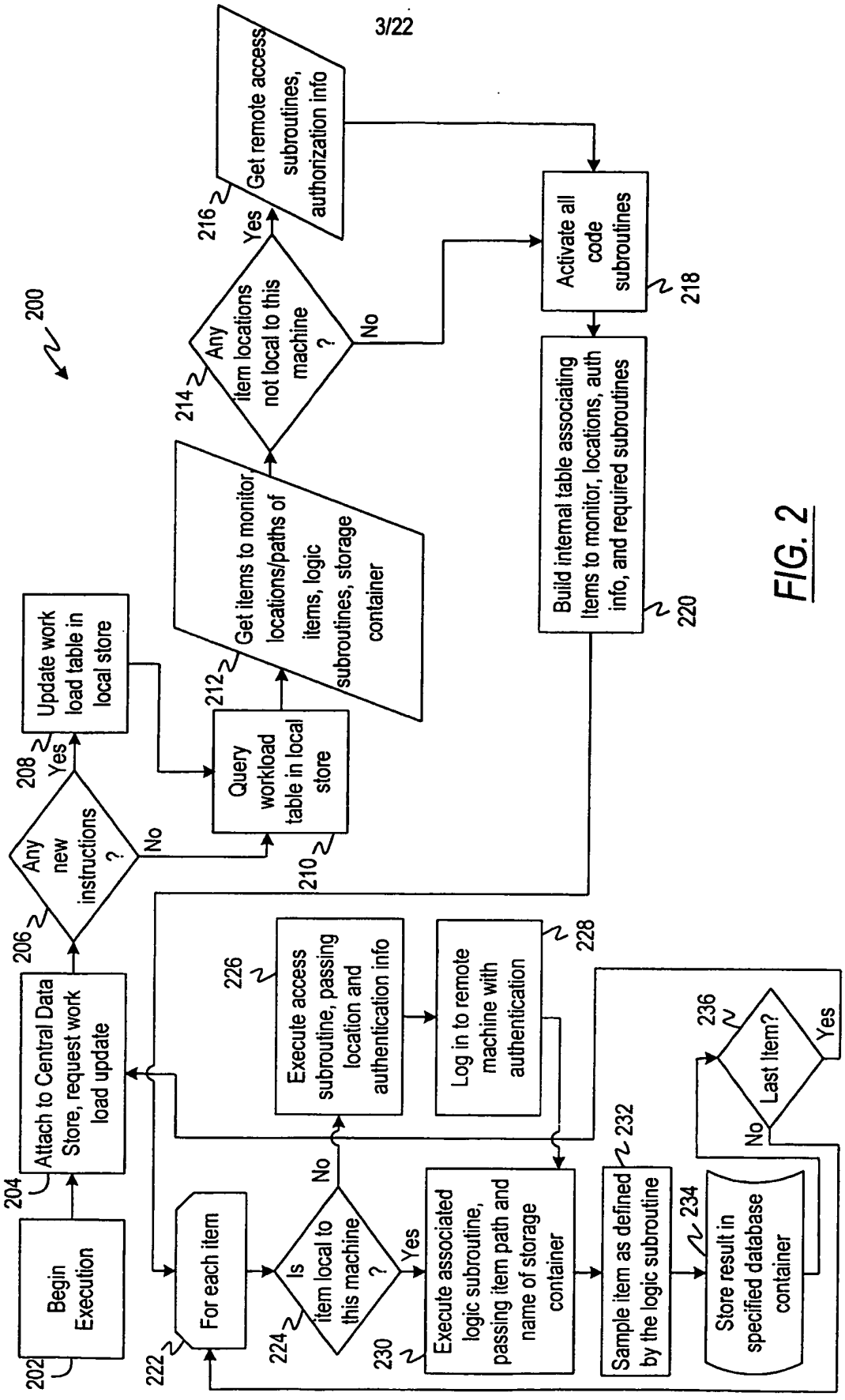

200
Begin Execution
202
Attach to Central Data Store, request work load update
204
For each item
222
Is item local to this machine ?
224
No
Yes
Execute associated logic subroutine, passing item path and name of storage container
230
Sample item as defined by the logic subroutine
232
Store result in specified database container
234
Last Item?
236
No
Yes
Execute access subroutine, passing location and authentication info
226
Log in to remote machine with authentication
228
Any new instructions ?
206
No
Yes
208
Update work load table in local store
Query workload table in local store
210
Get items to monitor locations/paths of items, logic subroutines, storage container
212
Any item locations not local to this machine ?
214
No
Yes
216
Get remote access subroutines, authorization info
Activate all code subroutines
218
Build internal table associating Items to monitor, locations, auth info, and required subroutines
220

FIG. 2

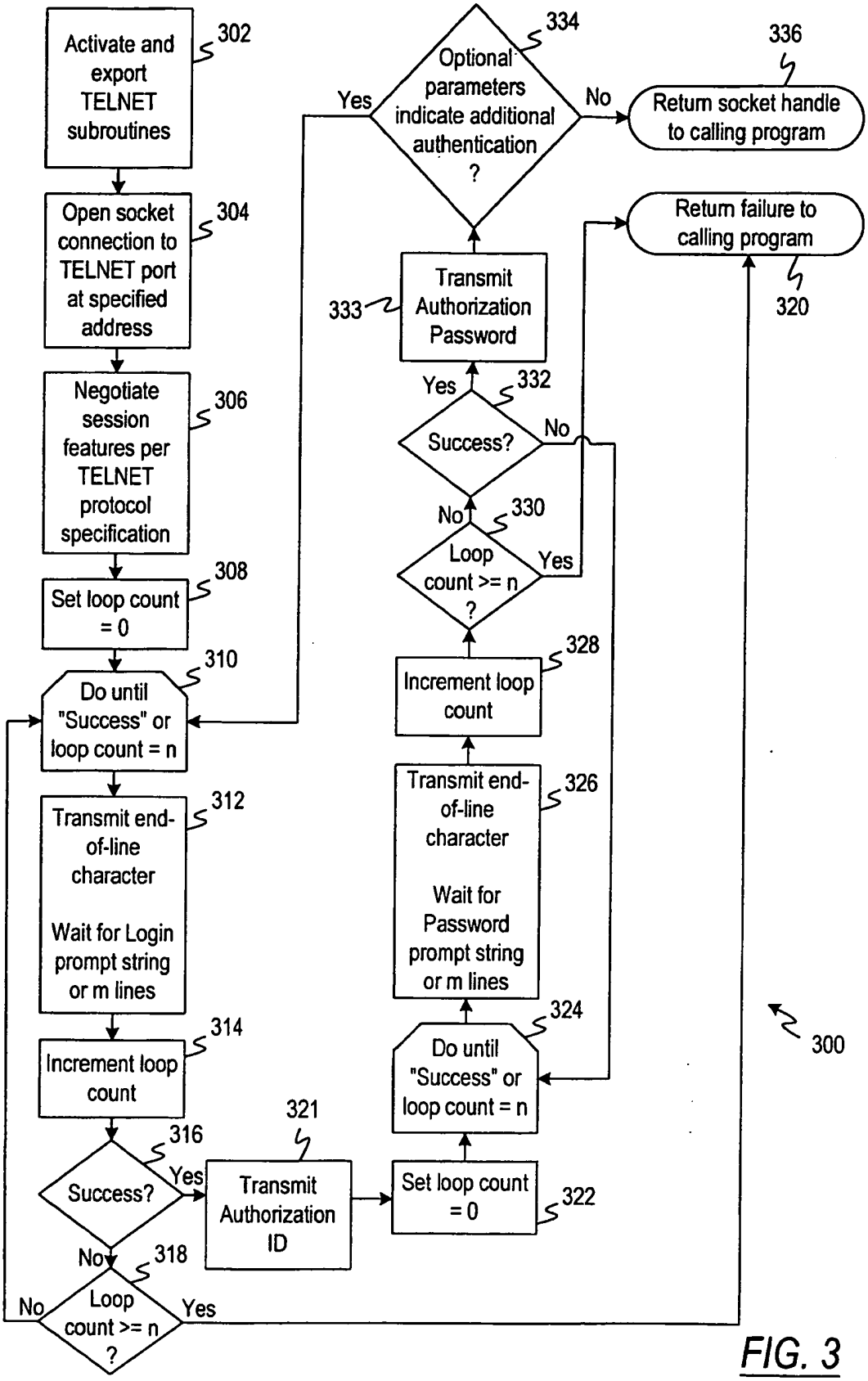

Activate and export TELNET subroutines
302
Open socket connection to TELNET port at specified address
304
Negotiate session features per TELNET protocol specification
306
Set loop count = 0
308
Do until "Success" or loop count = n
310
Transmit end-of-line character
Wait for Login prompt string or m lines
312
Increment loop count
314
Success?
316
Yes
No
Loop count >= n ?
318
No
Yes
Transmit Authorization ID
321
Set loop count = 0
322
Do until "Success" or loop count = n
324
Transmit end-of-line character
Wait for Password prompt string or m lines
326
Increment loop count
328
Loop count >= n ?
330
No
Yes
Success?
332
Yes
No
Transmit Authorization Password
333
Optional parameters indicate additional authentication ?
334
Yes
No
Return socket handle to calling program
336
Return failure to calling program
320
300

FIG. 3

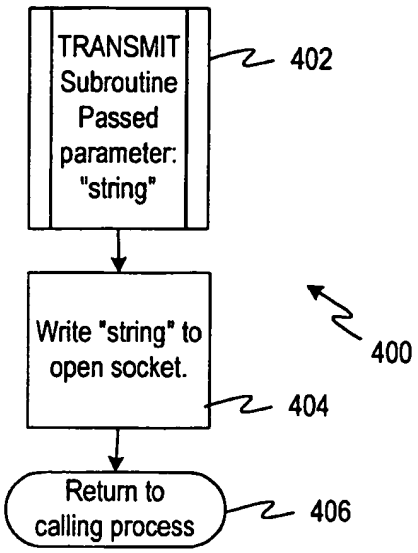

TRANSMIT Subroutine Passed parameter: "string"
402
Write "string" to open socket.
400
404
Return to calling process
406

FIG. 4

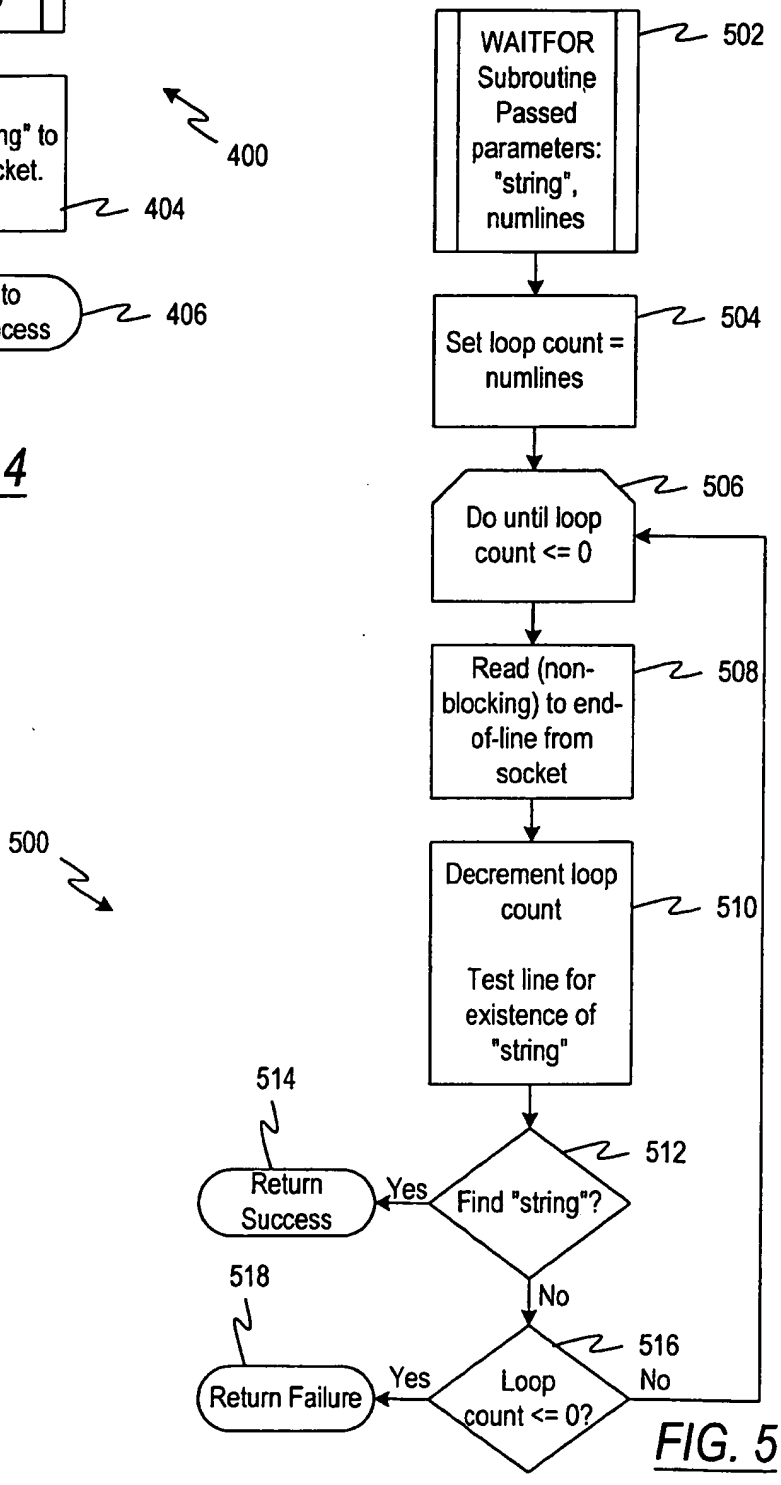

WAITFOR Subroutine Passed parameters: "string", numlines
502
Set loop count = numlines
504
Do until loop count <= 0
506
Read (non-blocking) to end-of-line from socket
508
500
Decrement loop count
Test line for existence of "string"
510
514
Return Success
Yes
Find "string"?
512
No
518
Return Failure
Yes
Loop count <= 0?
516
No

FIG. 5

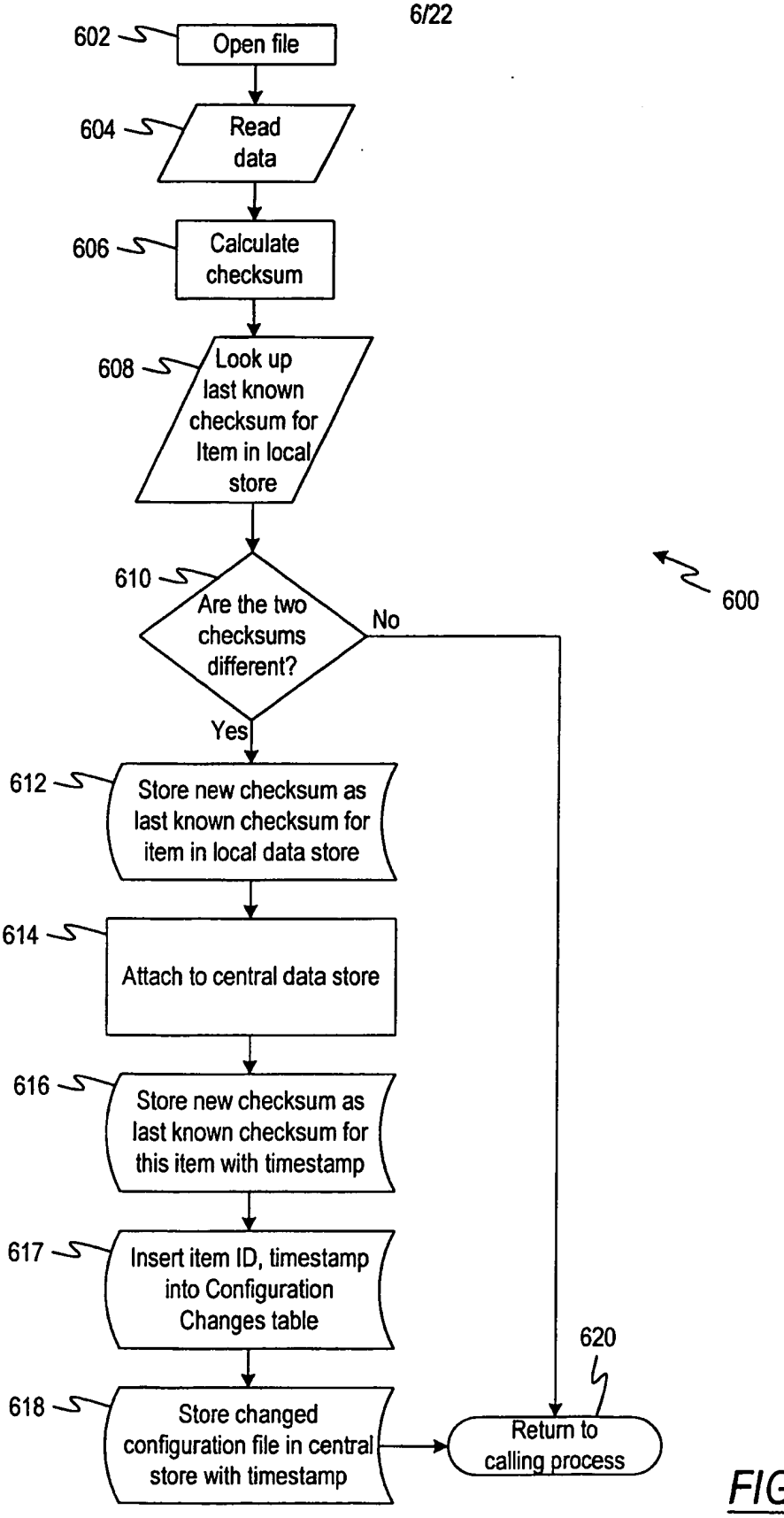

602
Open file
604
Read data
606
Calculate checksum
608
Look up last known checksum for Item in local store
610
Are the two checksums different?
No
Yes
600
612
Store new checksum as last known checksum for item in local data store
614
Attach to central data store
616
Store new checksum as last known checksum for this item with timestamp
617
Insert item ID, timestamp into Configuration Changes table
618
Store changed configuration file in central store with timestamp
620
Return to calling process

FIG. 6

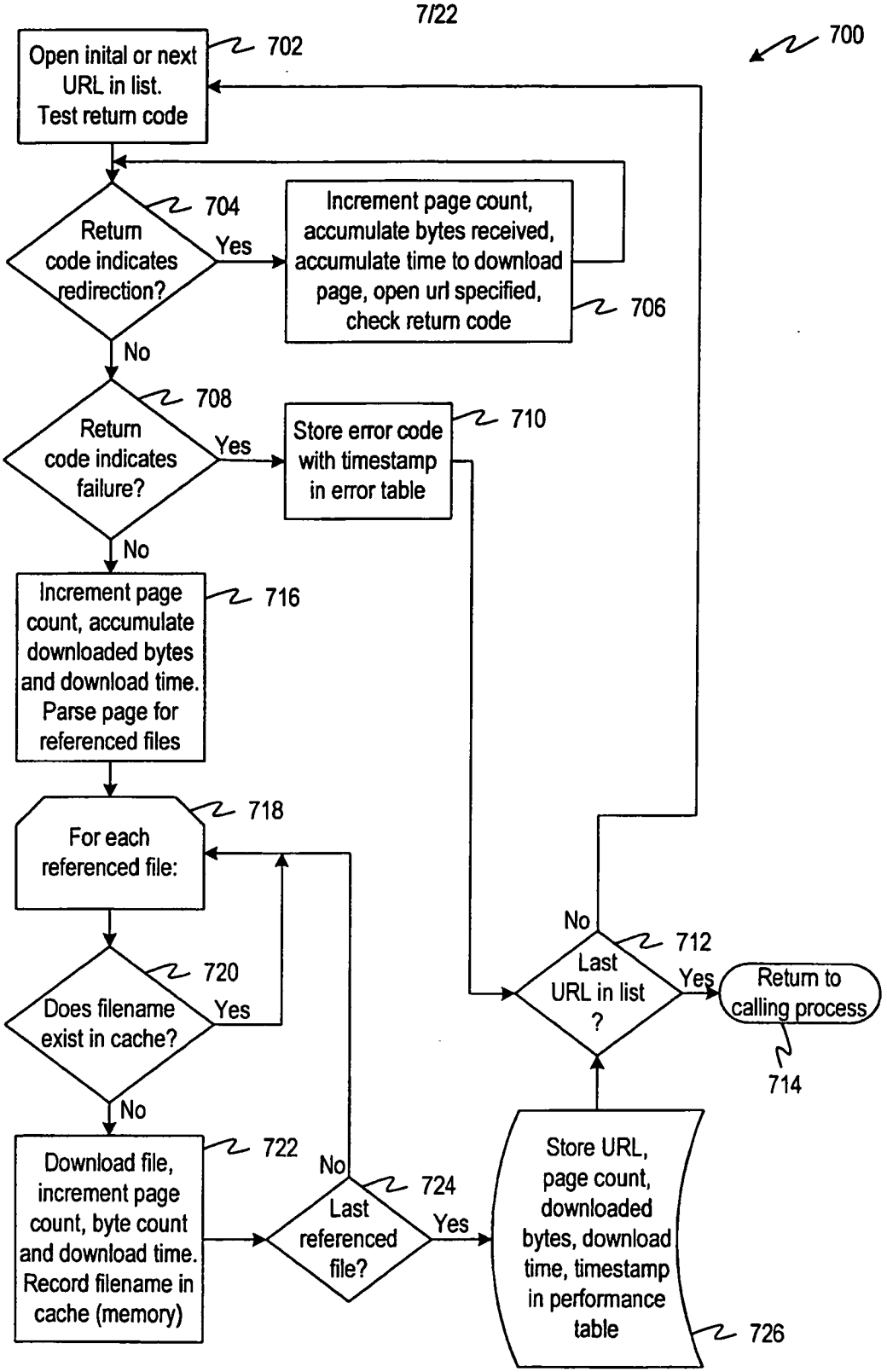

700
Open inital or next URL in list. Test return code
702
Return code indicates redirection?
704
Yes
Increment page count, accumulate bytes received, accumulate time to download page, open url specified, check return code
706
No
Return code indicates failure?
708
Yes
Store error code with timestamp in error table
710
No
Increment page count, accumulate downloaded bytes and download time. Parse page for referenced files
716
For each referenced file:
718
Does filename exist in cache?
720
Yes
No
Download file, increment page count, byte count and download time. Record filename in cache (memory)
722
Last referenced file?
724
No
Yes
Store URL, page count, downloaded bytes, download time, timestamp in performance table
726
Last URL in list ?
712
No
Yes
Return to calling process
714

FIG. 7

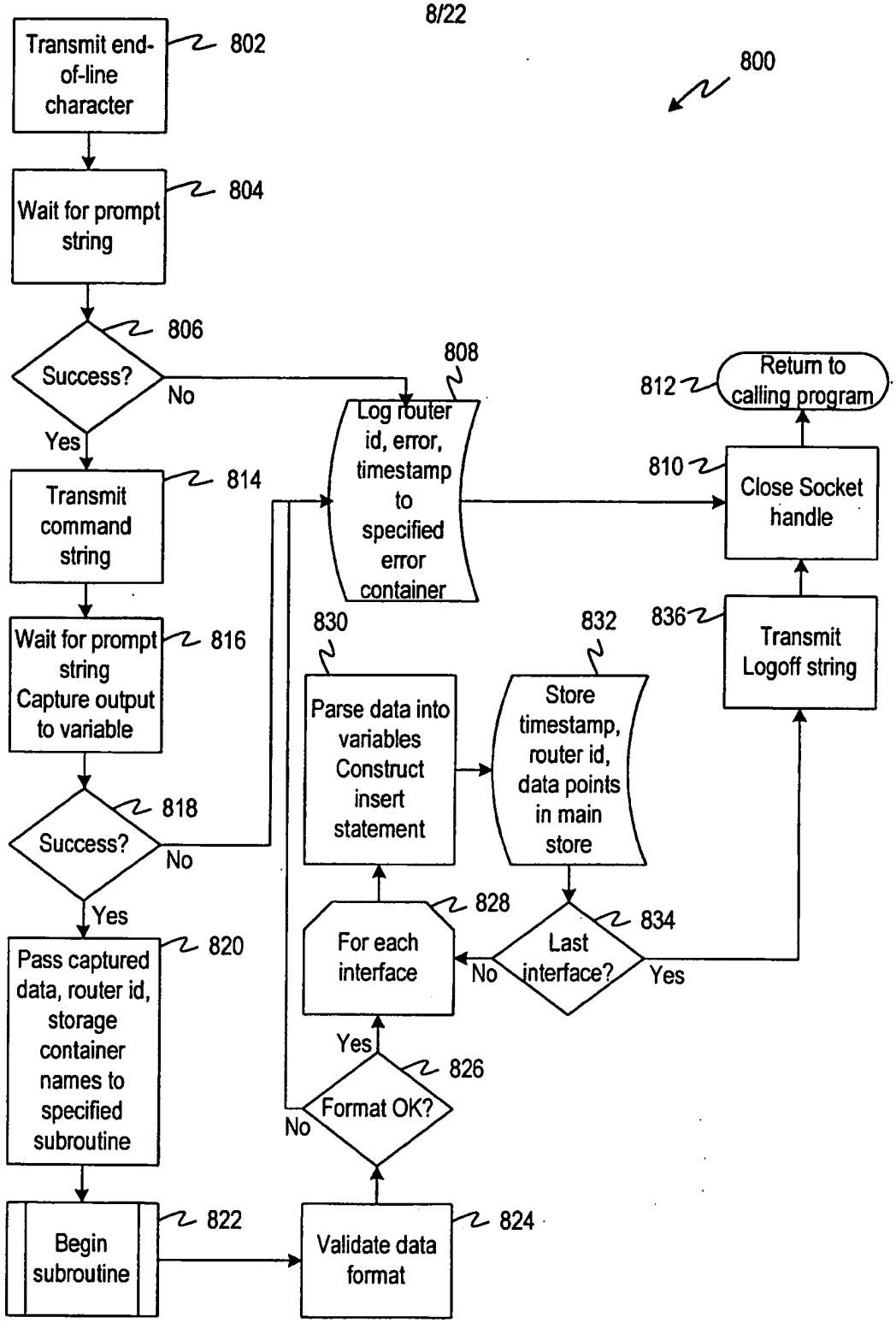

800
Transmit end-of-line character
802
Wait for prompt string
804
Success?
806
No
Yes
Transmit command string
814
Wait for prompt string Capture output to variable
816
Success?
818
No
Yes
Pass captured data, router id, storage container names to specified subroutine
820
Begin subroutine
822
Validate data format
824
Format OK?
826
No
Yes
For each interface
828
Parse data into variables Construct insert statement
830
Store timestamp, router id, data points in main store
832
Last interface?
834
No
Yes
Log router id, error, timestamp to specified error container
808
Close Socket handle
810
Return to calling program
812
Transmit Logoff string
836

FIG. 8

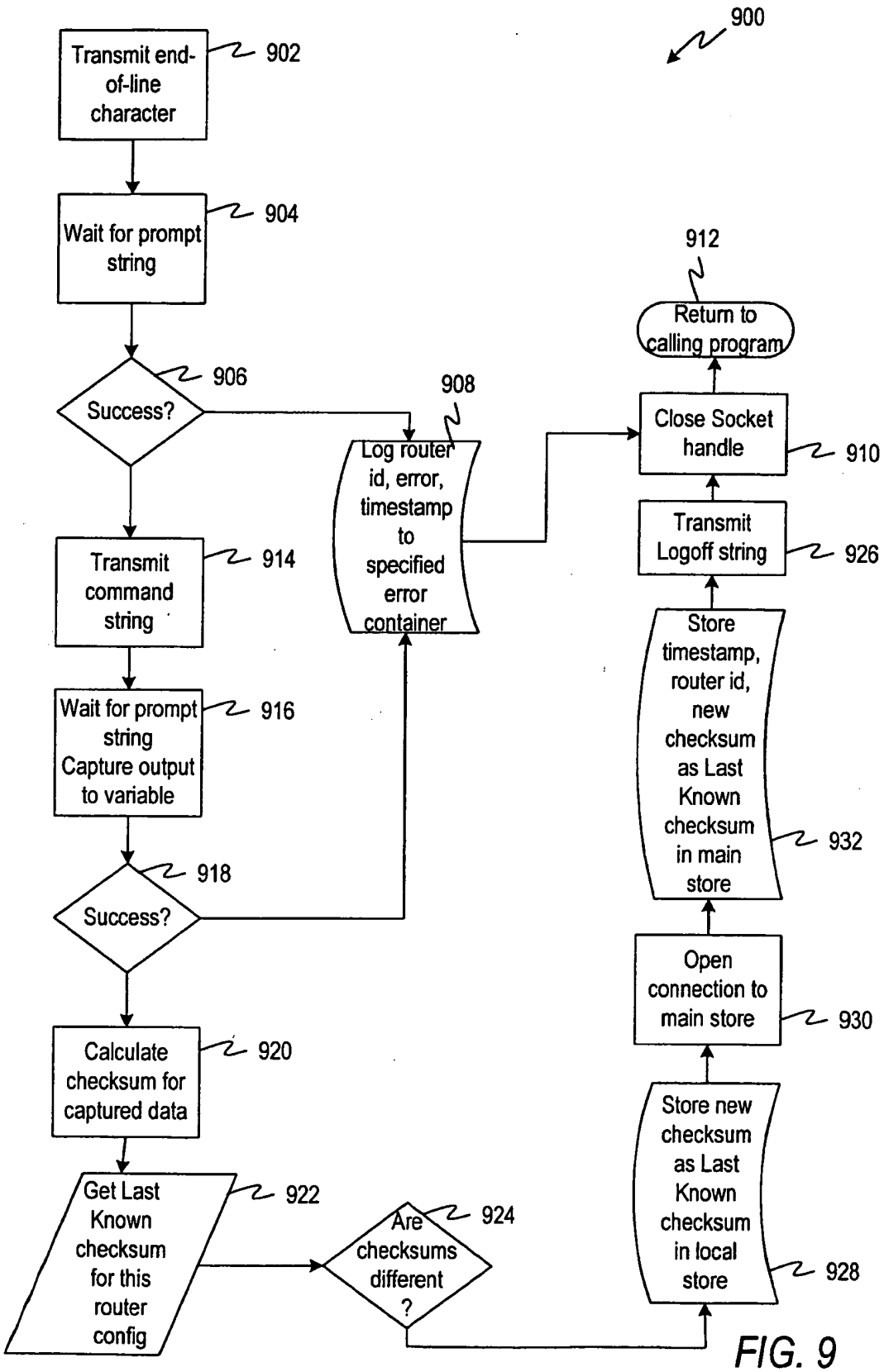

900
Transmit end-of-line character
902
Wait for prompt string
904
Success?
906
Transmit command string
914
Wait for prompt string Capture output to variable
916
Success?
918
Calculate checksum for captured data
920
Get Last Known checksum for this router config
922
Are checksums different ?
924
Log router id, error, timestamp to specified error container
908
Close Socket handle
910
Return to calling program
912
Transmit Logoff string
926
Store timestamp, router id, new checksum as Last Known checksum in main store
932
Open connection to main store
930
Store new checksum as Last Known checksum in local store
928

FIG. 9

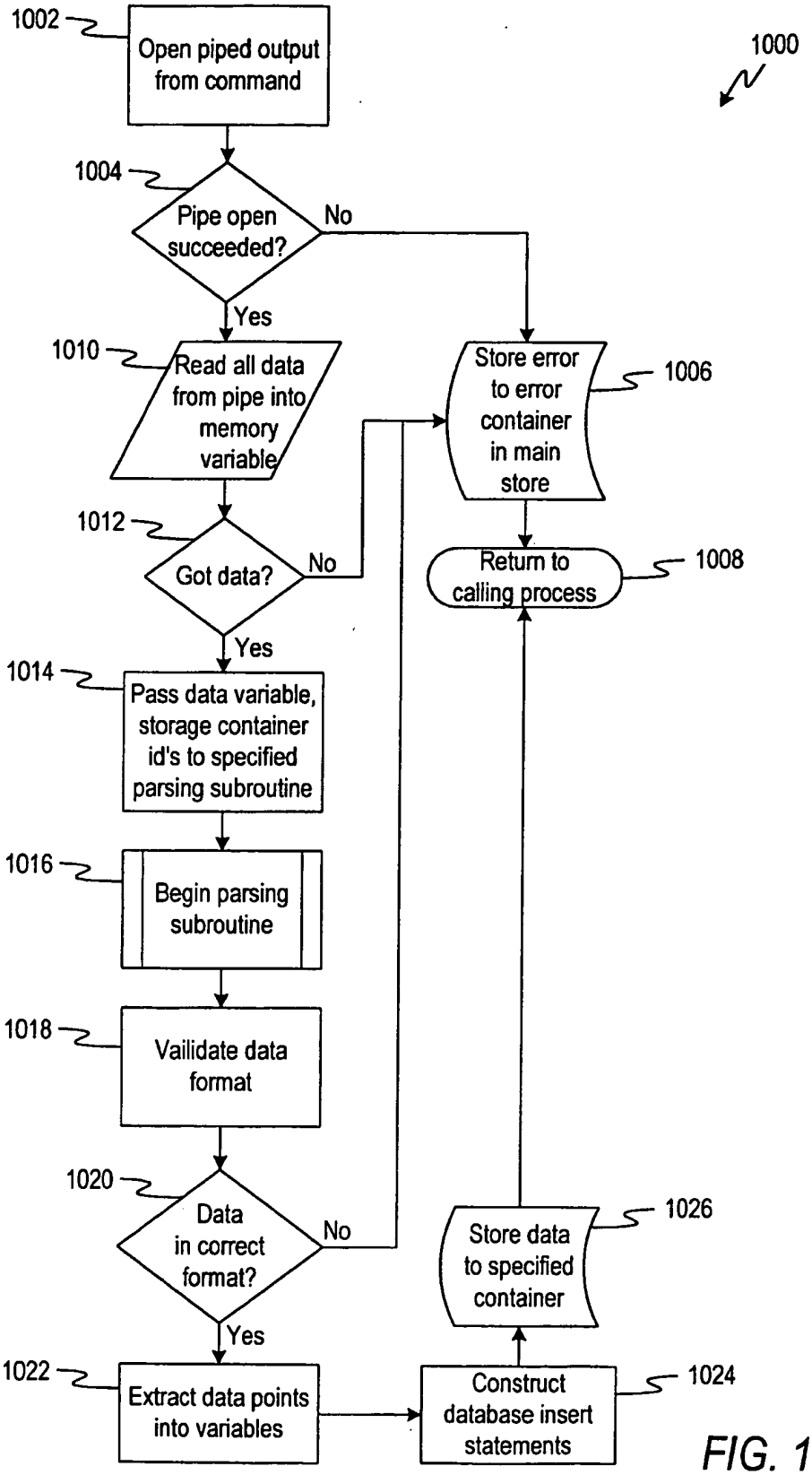

1000
1002
Open piped output from command
1004
Pipe open succeeded?
No
Yes
1010
Read all data from pipe into memory variable
1012
Got data?
No
Yes
1014
Pass data variable, storage container id's to specified parsing subroutine
1016
Begin parsing subroutine
1018
Vailidate data format
1020
Data in correct format?
No
Yes
1022
Extract data points into variables
1024
Construct database insert statements
1026
Store data to specified container
1006
Store error to error container in main store
1008
Return to calling process

FIG. 10

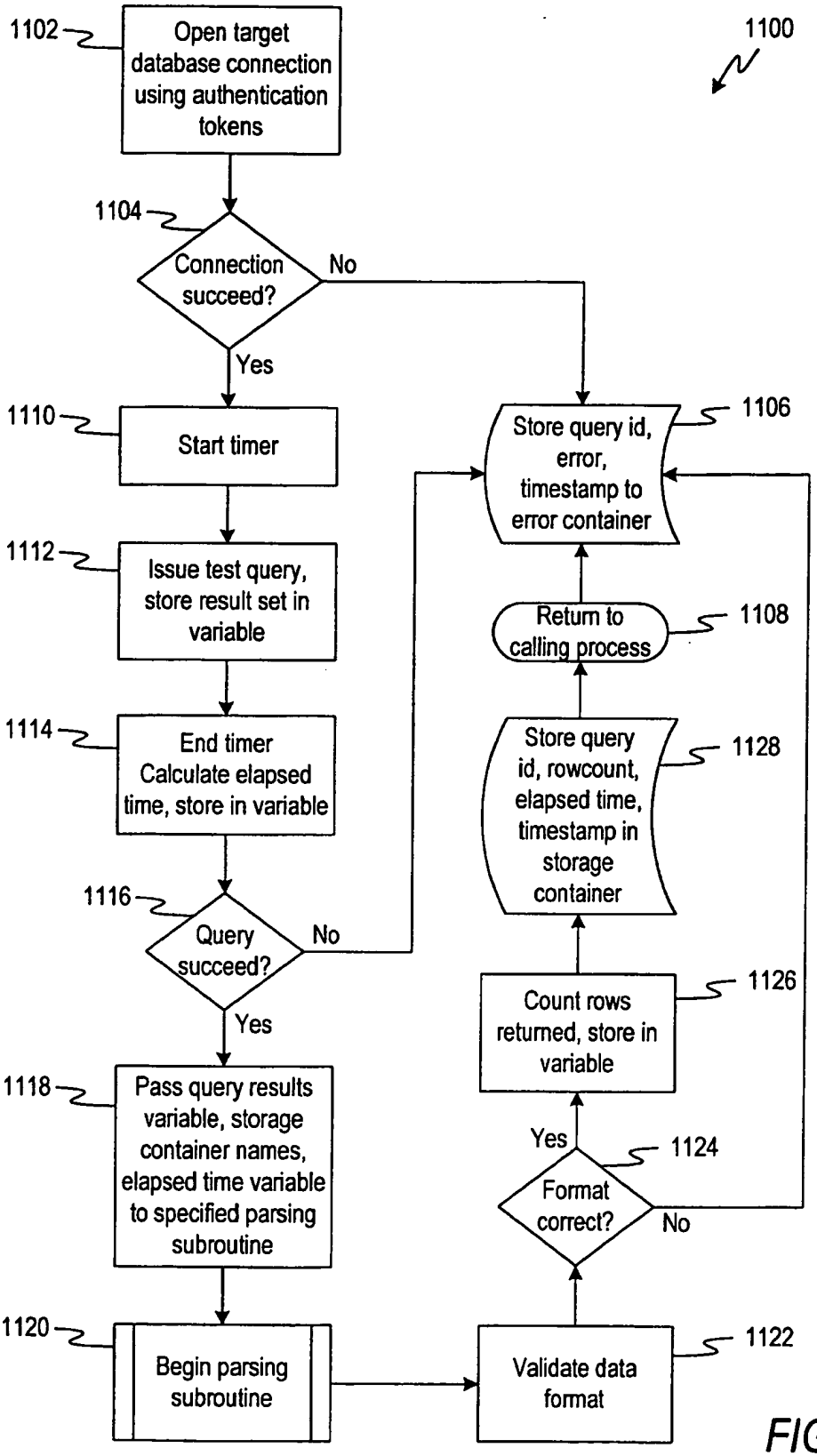

1100
1102
Open target database connection using authentication tokens
1104
Connection succeed?
No
Yes
1110
Start timer
1112
Issue test query, store result set in variable
1114
End timer Calculate elapsed time, store in variable
1116
Query succeed?
No
Yes
1118
Pass query results variable, storage container names, elapsed time variable to specified parsing subroutine
1120
Begin parsing subroutine
1122
Validate data format
1124
Format correct?
Yes
No
1126
Count rows returned, store in variable
1128
Store query id, rowcount, elapsed time, timestamp in storage container
1108
Return to calling process
1106
Store query id, error, timestamp to error container

FIG. 11

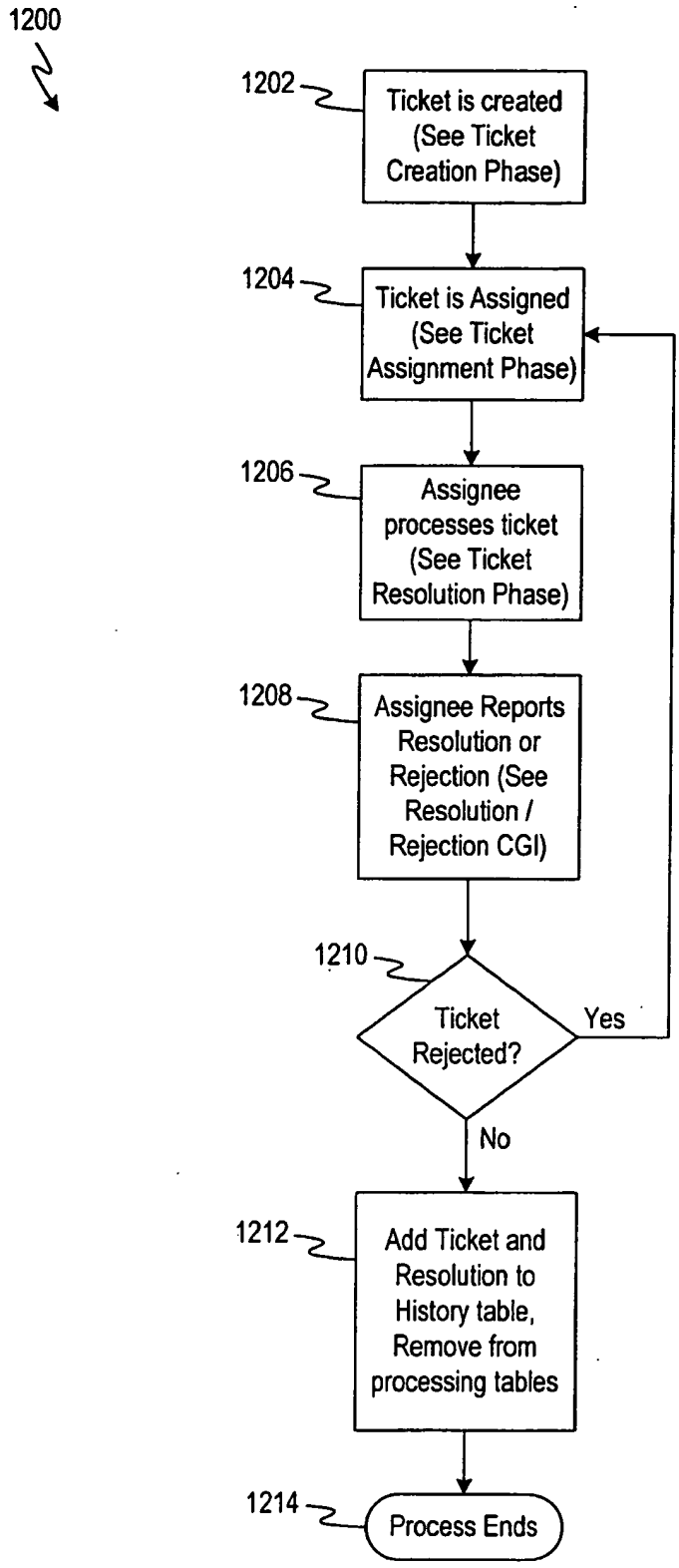

1200
1202
Ticket is created (See Ticket Creation Phase)
1204
Ticket is Assigned (See Ticket Assignment Phase)
1206
Assignee processes ticket (See Ticket Resolution Phase)
1208
Assignee Reports Resolution or Rejection (See Resolution / Rejection CGI)
1210
Ticket Rejected?
Yes
No
1212
Add Ticket and Resolution to History table, Remove from processing tables
1214
Process Ends

FIG. 12

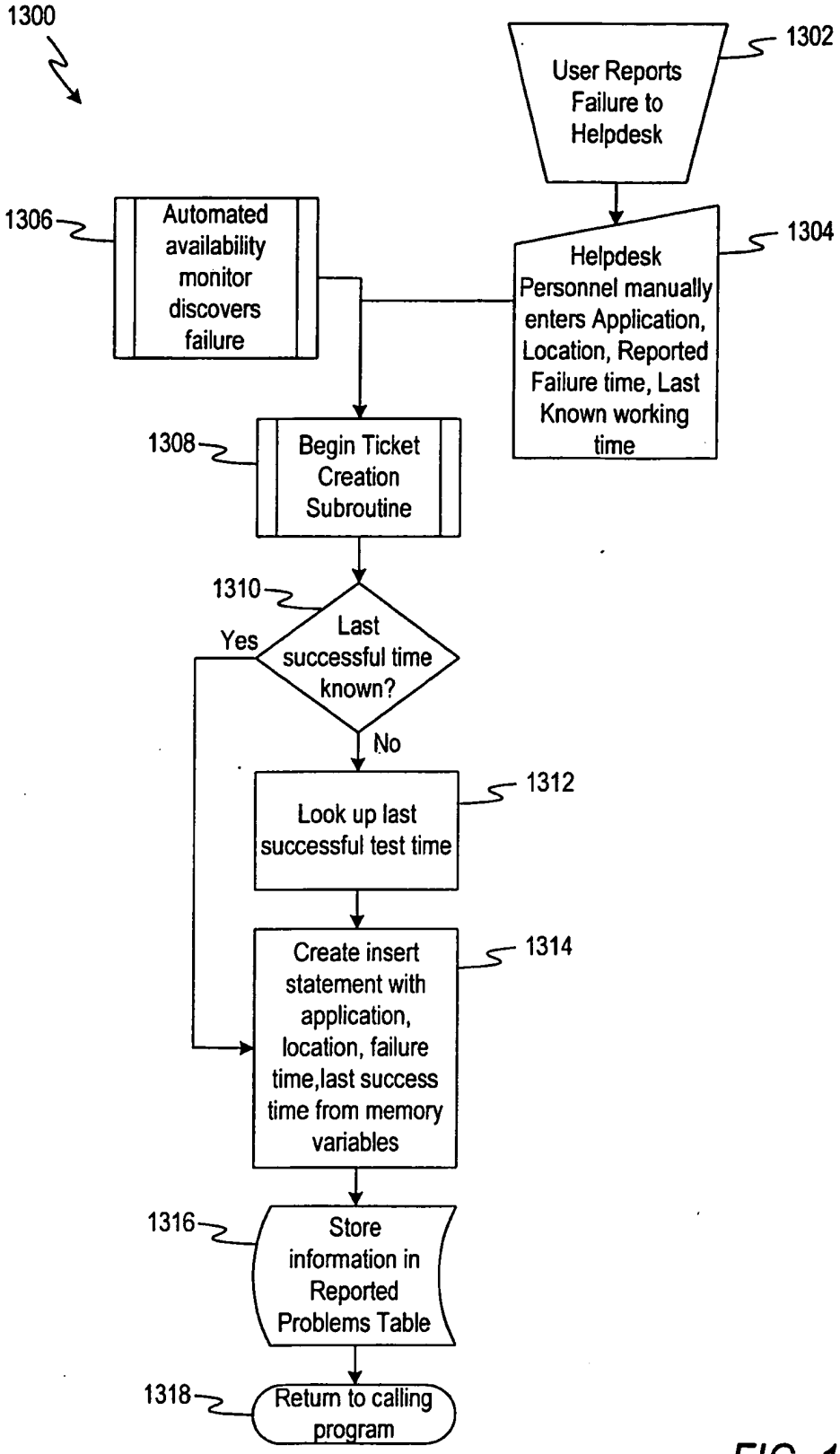

1300
User Reports Failure to Helpdesk
1302
1306
Automated availability monitor discovers failure
Helpdesk Personnel manually enters Application, Location, Reported Failure time, Last Known working time
1304
1308
Begin Ticket Creation Subroutine
1310
Last successful time known?
Yes
No
1312
Look up last successful test time
Create insert statement with application, location, failure time,last success time from memory variables
1314
1316
Store information in Reported Problems Table
1318
Return to calling program

FIG. 13

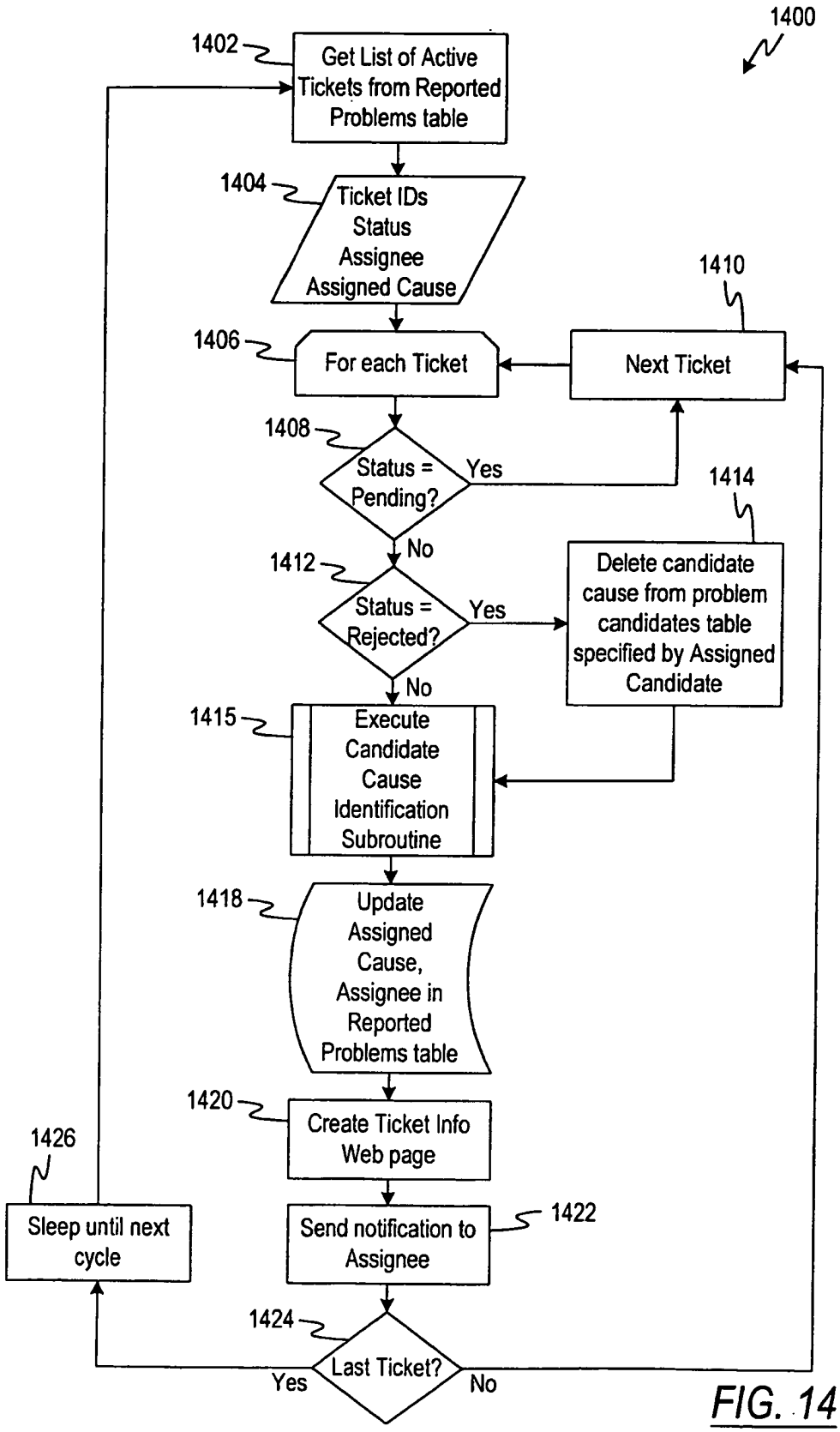

1400
1402
Get List of Active Tickets from Reported Problems table
1404
Ticket IDs
Status
Assignee
Assigned Cause
1410
1406
For each Ticket
Next Ticket
1408
Status = Pending?
Yes
1414
No
1412
Status = Rejected?
Yes
Delete candidate cause from problem candidates table specified by Assigned Candidate
No
1415
Execute Candidate Cause Identification Subroutine
1418
Update Assigned Cause, Assignee in Reported Problems table
1420
Create Ticket Info Web page
1426
Sleep until next cycle
Send notification to Assignee
1422
1424
Last Ticket?
Yes
No

FIG. 14

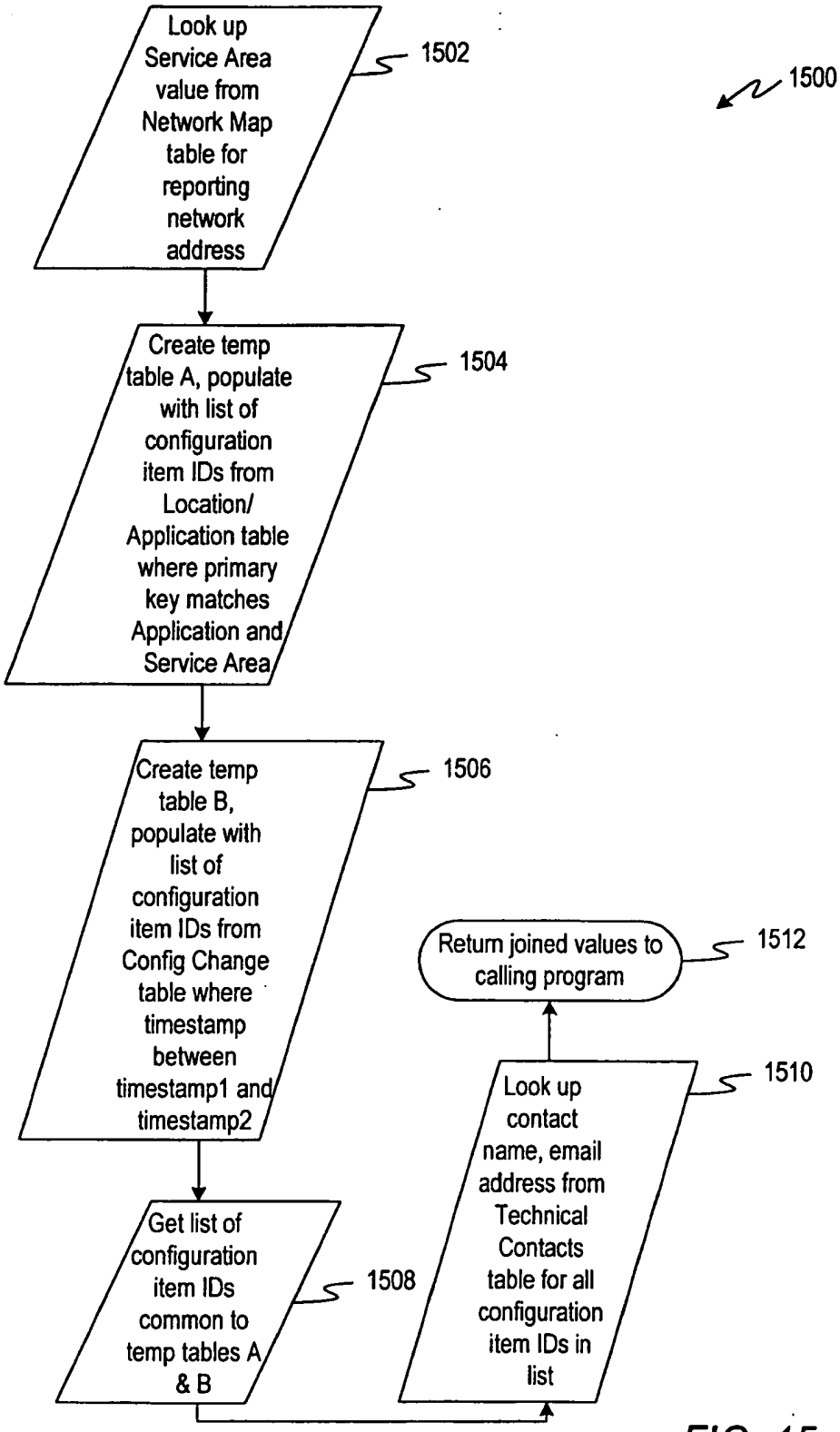

Look up Service Area value from Network Map table for reporting network address
1502
1500
Create temp table A, populate with list of configuration item IDs from Location/ Application table where primary key matches Application and Service Area
1504
Create temp table B, populate with list of configuration item IDs from Config Change table where timestamp between timestamp1 and timestamp2
1506
Return joined values to calling program
1512
Look up contact name, email address from Technical Contacts table for all configuration item IDs in list
1510
Get list of configuration item IDs common to temp tables A & B
1508

FIG. 15

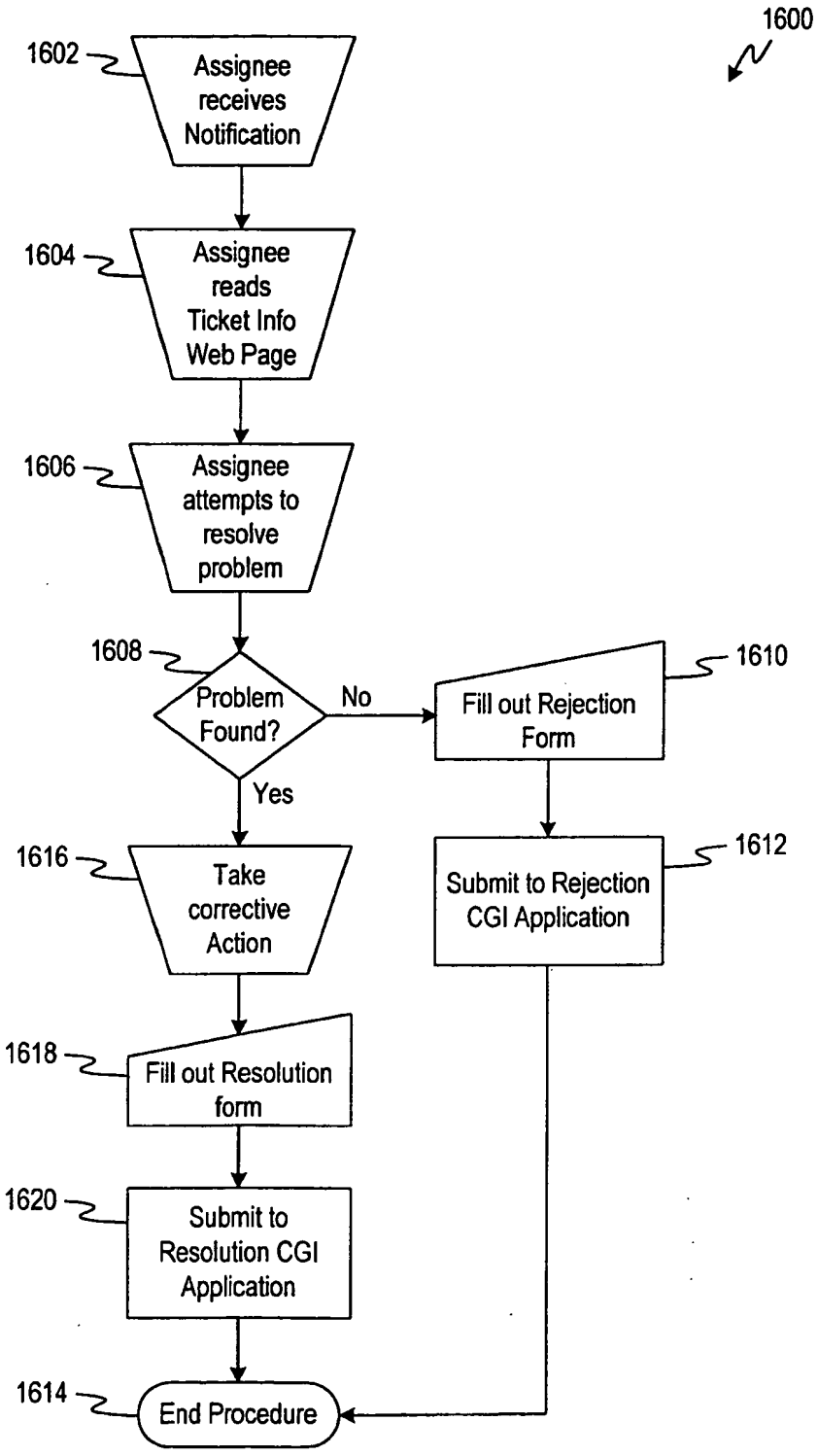

1600
1602
Assignee receives Notification
1604
Assignee reads Ticket Info Web Page
1606
Assignee attempts to resolve problem
1608
Problem Found?
No
Yes
1610
Fill out Rejection Form
1612
Submit to Rejection CGI Application
1616
Take corrective Action
1618
Fill out Resolution form
1620
Submit to Resolution CGI Application
1614
End Procedure

FIG. 16

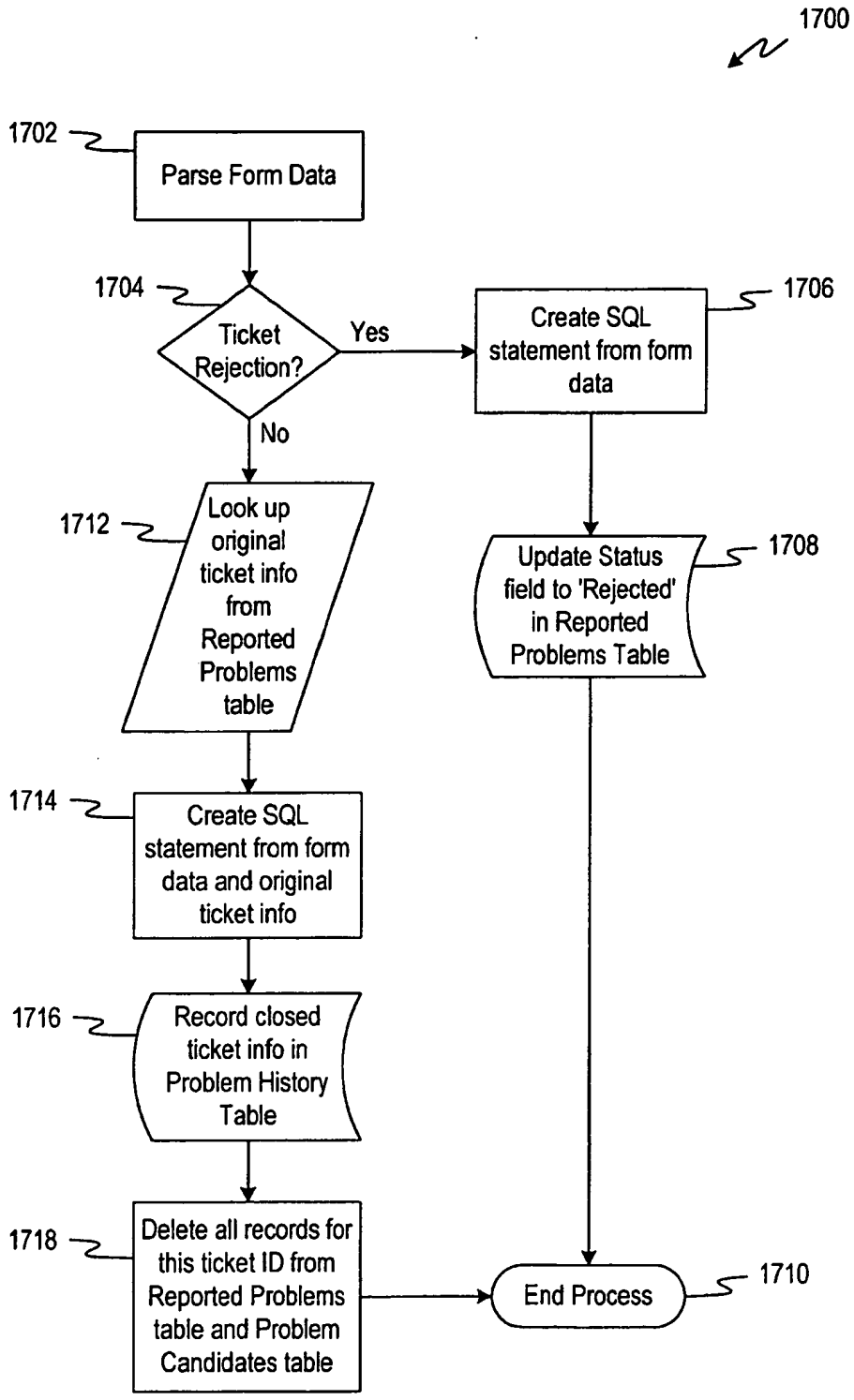

1700
1702
Parse Form Data
1704
Ticket Rejection?
Yes
No
1706
Create SQL statement from form data
1708
Update Status field to 'Rejected' in Reported Problems Table
1712
Look up original ticket info from Reported Problems table
1714
Create SQL statement from form data and original ticket info
1716
Record closed ticket info in Problem History Table
1718
Delete all records for this ticket ID from Reported Problems table and Problem Candidates table
1710
End Process

FIG. 17

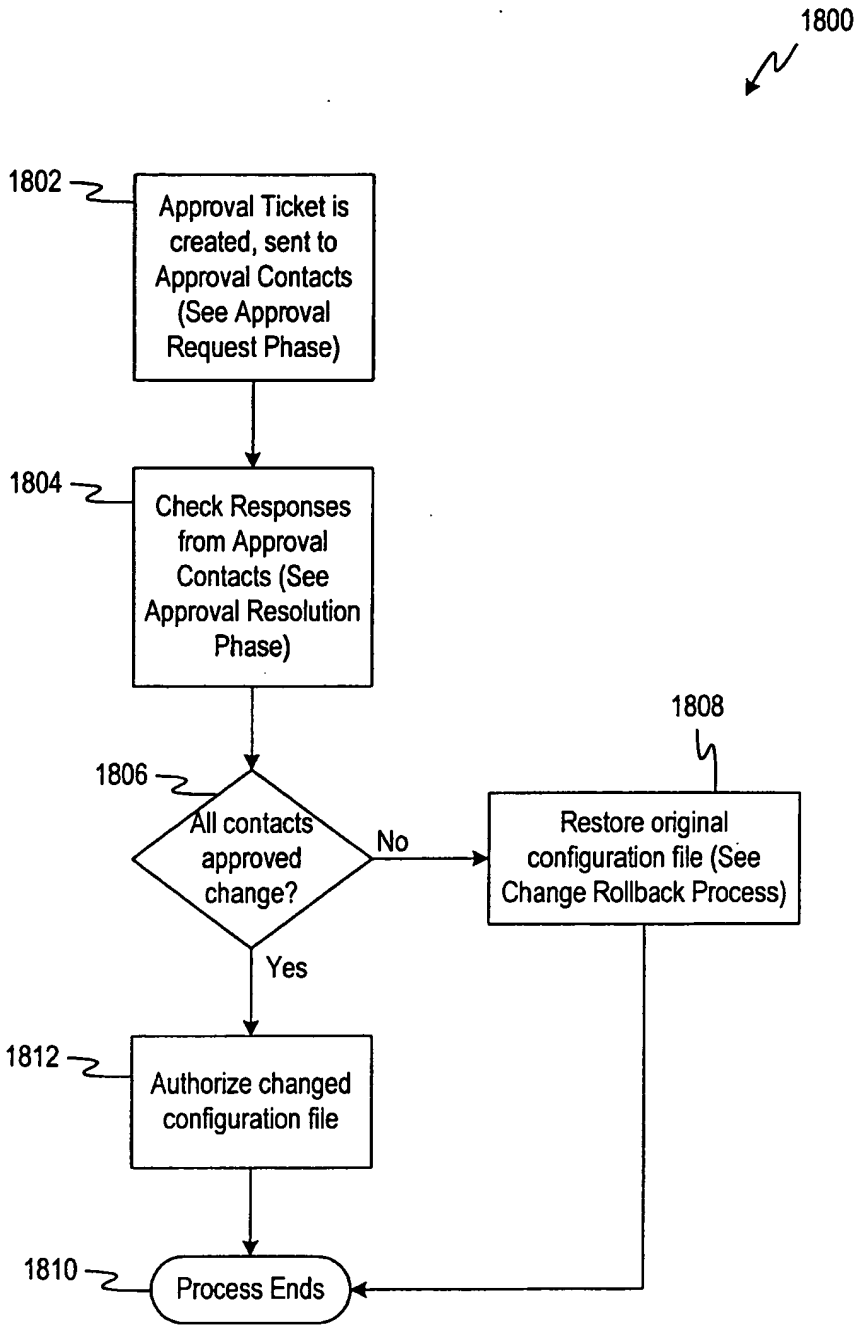

1800
1802
Approval Ticket is created, sent to Approval Contacts (See Approval Request Phase)
1804
Check Responses from Approval Contacts (See Approval Resolution Phase)
1808
1806
All contacts approved change?
No
Restore original configuration file (See Change Rollback Process)
Yes
1812
Authorize changed configuration file
1810
Process Ends

FIG. 18

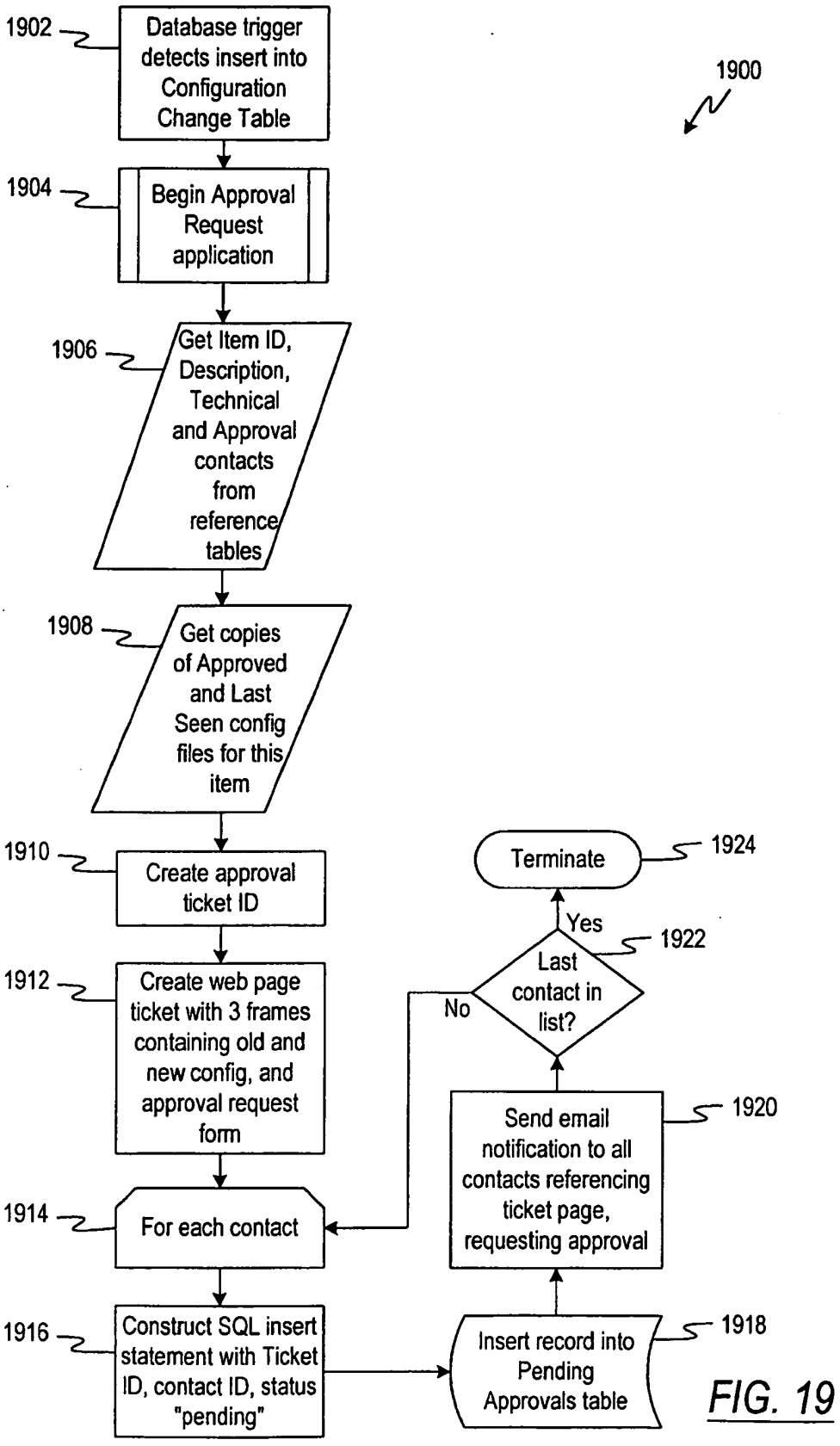

1900
1902
Database trigger detects insert into Configuration Change Table
1904
Begin Approval Request application
1906
Get Item ID, Description, Technical and Approval contacts from reference tables
1908
Get copies of Approved and Last Seen config files for this item
1910
Create approval ticket ID
1912
Create web page ticket with 3 frames containing old and new config, and approval request form
1914
For each contact
1916
Construct SQL insert statement with Ticket ID, contact ID, status "pending"
1918
Insert record into Pending Approvals table
1920
Send email notification to all contacts referencing ticket page, requesting approval
1922
Last contact in list?
Yes
No
1924
Terminate

FIG. 19

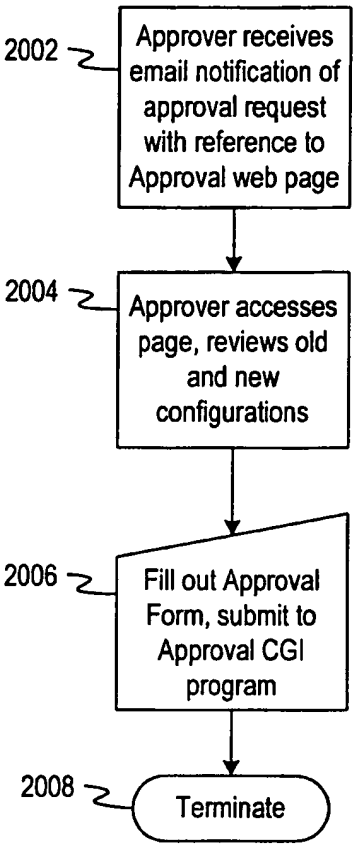

2000
2002
Approver receives email notification of approval request with reference to Approval web page
2004
Approver accesses page, reviews old and new configurations
2006
Fill out Approval Form, submit to Approval CGI program
2008
Terminate

FIG. 20

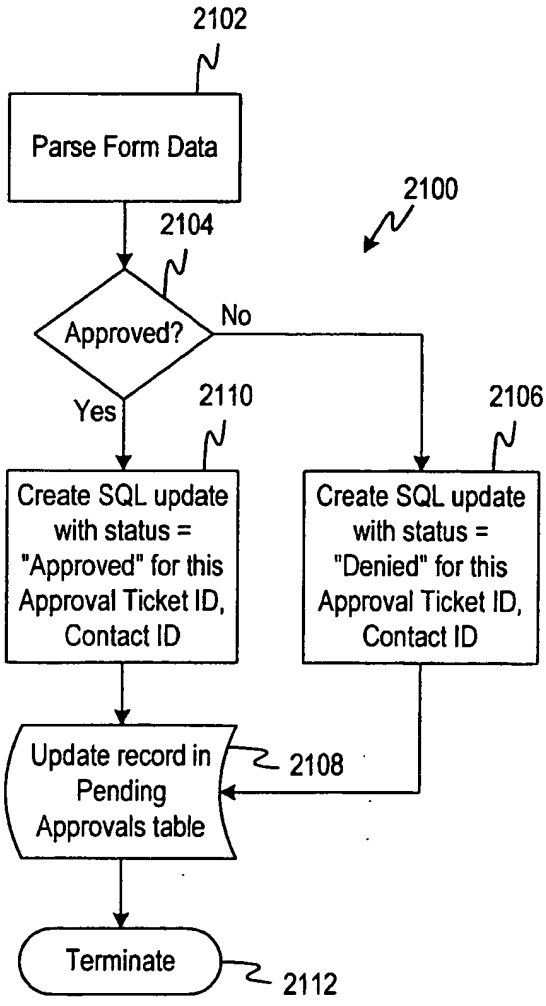

2102
Parse Form Data
2100
2104
Approved?
No
Yes
2110
2106
Create SQL update with status = "Approved" for this Approval Ticket ID, Contact ID
Create SQL update with status = "Denied" for this Approval Ticket ID, Contact ID
Update record in Pending Approvals table
2108
Terminate
2112

FIG. 21

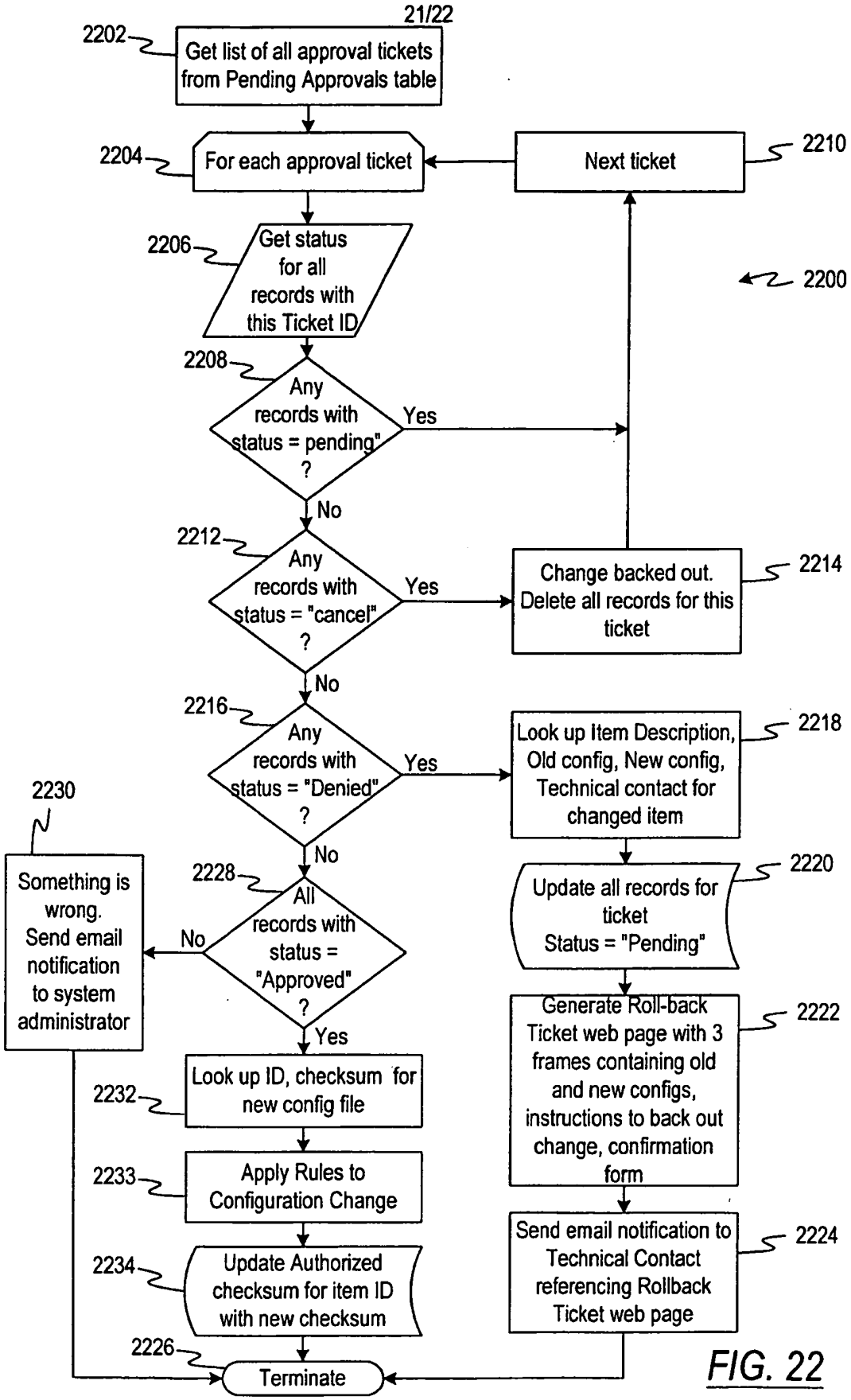

2202
Get list of all approval tickets from Pending Approvals table
2204
For each approval ticket
Next ticket
2210
2206
Get status for all records with this Ticket ID
2200
2208
Any records with status = pending" ?
Yes
No
2212
Any records with status = "cancel" ?
Yes
Change backed out. Delete all records for this ticket
2214
No
2216
Any records with status = "Denied" ?
Yes
Look up Item Description, Old config, New config, Technical contact for changed item
2218
2230
No
2228
All records with status = "Approved" ?
No
Something is wrong. Send email notification to system administrator
Yes
Update all records for ticket Status = "Pending"
2220
Generate Roll-back Ticket web page with 3 frames containing old and new configs, instructions to back out change, confirmation form
2222
2232
Look up ID, checksum for new config file
2233
Apply Rules to Configuration Change
2234
Update Authorized checksum for item ID with new checksum
Send email notification to Technical Contact referencing Rollback Ticket web page
2224
2226
Terminate

FIG. 22

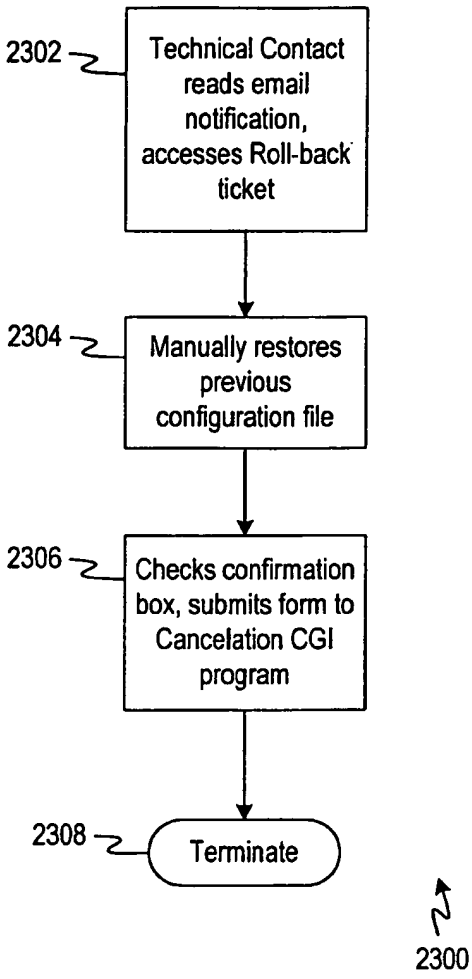

2302
Technical Contact reads email notification, accesses Roll-back ticket
2304
Manually restores previous configuration file
2306
Checks confirmation box, submits form to Cancelation CGI program
2308
Terminate
2300

FIG. 23

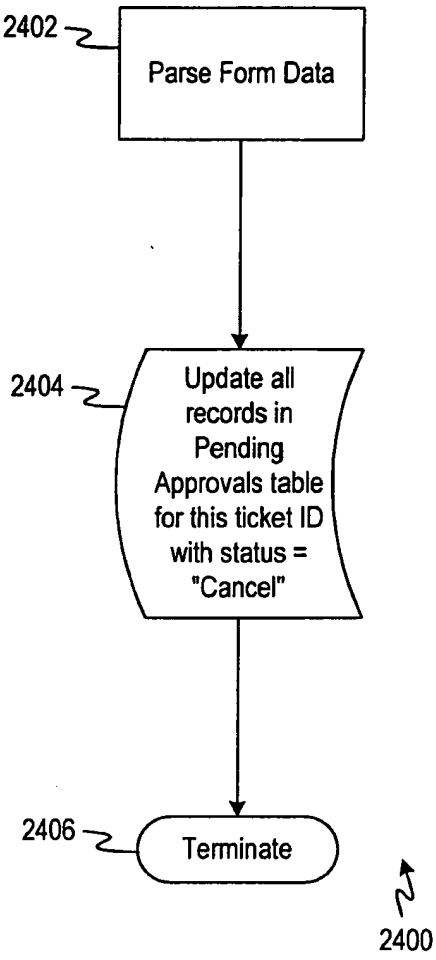

2402
Parse Form Data
2404
Update all records in Pending Approvals table for this ticket ID with status = "Cancel"
2406
Terminate
2400

FIG. 24

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/00615

A. CLASSIFICATI N OF SUBJECT MATTER
IPC(7) : G06F 13/00, 17/00
US CL : 707/2, 4, 5 , 6, 102, 104; 711/102
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 707/2, 4, 5 , 6, 102, 104; 711/102

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

IEEE, Microsoft's Computer Dictionary

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

West, STN
Search terms: client, server, web server, function module, documentation, change, data collection, track.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y, P	US 5,884,298 A (SMITH II et al) 16 March 1999, abstract, col.4, lines 30-65, col.10 line 36 to col.11 line 38, col.14 lines 3-47, col.16 lines 11-35.	1-5
Y, P	US 5,905,981 A (LAWLER) 18 May 1999, abstract, figs.3 and 4, col.2 lines 14-53, col.4 lines 21-67.	1-5

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"B"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 10 MARCH 2000	Date of mailing of the international search report 26 APR 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer AHMAD MATAR Telephone No. (703) 305-4731

Form PCT/ISA/210 (second sheet) (July 1998)*